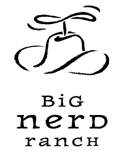

ПИТЕР®

Android Programming

The Big Nerd Ranch Guide

Bill Phillips & Brian Hardy



ПРОГРАММИРОВАНИЕ под Android

Брайан Харди, Билл Филлипс



Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2014

Б. Харди, Б. Филлипс

Программирование под Android. Для профессионалов

Перевел с английского Е. Матвеев

Заведующий редакцией	*А. Кривцов*
Руководитель проекта	*А. Юрченко*
Ведущий редактор	*Ю. Сергиенко*
Художественный редактор	*Л. Адуевская*
Корректор	*В. Листова*
Верстка	*Л. Родионова*

ББК 32.973.2-018.2

УДК 004.451

Харди Б. , Филлипс Б.

Х20 Программирование под Android. Для профессионалов. — СПб.: Питер, 2014. — 592 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-496-00502-9

Эта книга научит вас всем тонкостям разработки приложений для мобильных устройств, работающих на популярной операционной системе Android. Основанное на учебном курсе известного образовательного IT-проекта Big Nerd Ranch, это издание содержит ключевые концепции программирования в Android, разъяснение API и множество примеров кода и упражнений.

В процессе работы с книгой читатель самостоятельно разработает восемь Android-приложений разной сложности и тематики, включая клиент для загрузки фотографий из базы Flickr, приложение, имитирующее телевизионный пульт дистанционного управления, а также сервис геолокации, отслеживающий перемещения пользователя по всему свету и отображающий их на карте. Все учебные приложения были спроектированы таким образом, чтобы продемонстрировать важные концепции и приемы программирования под Android и дать опыт их практического применения.

12+ (Для детей старше 12 лет. В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), д. 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.
Подписано в печать 30.10.13. Формат 70х100/16. Усл. п. л. 47,730. Тираж 1700. Заказ
Отпечатано в полном соответствии с качеством предоставленных издательством материалов
в ОГУП «Областная типография «Печатный двор». 432049, Ульяновск, ул. Пушкарева, д.27.

Содержание

Посвящается Доновану. Надеюсь, что его жизнь будет полна активностей и он всегда будет знать, когда применять фрагменты.

— Б. Х.

Благодарности

Нам немного неудобно, что на обложке книги указаны только наши имена. Дело в том, что без многочисленных помощников эта книга никогда бы не вышла в свет. Мы им всем многим обязаны.

- Крис Стюарт (Chris Stewart) и Оуэн Мэтьюз (Owen Matthews) предложили ряд замечательных идей по основополагающему материалу нескольких глав.
- Мы благодарны нашим коллегам-преподавателям Крису Стюарту и Кристоферу Муру (Christopher Moory) за их терпение по преподаванию промежуточных рабочих материалов, за их предложения и исправления по этим материалам и за консультации при планировании радикальных изменений.
- Наши коллеги Болот Керимбаев (Bolot Kerimbaev) и Эндрю Лунсфорд (Andrew Lunsford) высказали исключительно ценное мнение по поводу использования фрагментов.
- Технические рецензенты Фрэнк Роблес (Frank Robles), Джим Стил (Jim Steele), Лора Касселл (Laura Cassell), Марк Далримпл (Mark Dalrymple) и Магнус Дал (Magnus Dahl) помогли нам найти и исправить неточности в тексте.
- Спасибо Аарону Хиллегассу (Aaron Hillegass). Вера Аарона в людей — одна из великих, ужасающих сил природы. Без нее мы бы никогда не получили возможности написать эту книгу и не дописали бы ее до конца. (А еще он нам платил, что было очень любезно с его стороны.)
- Наш редактор Сьюзен Лопер (Susan Loper) обладает невероятным умением превращать наши разглагольствования и несмешные шутки в продуманный, лаконичный текст. Без ее помощи читать эту книгу было бы не так приятно. Она научила нас всему, что мы знаем о четком и доступном изложении технического материала.
- Спасибо NASA. Наша книга кажется маленькой и глупой по сравнению с их усилиями по изучению Солнечной системы.

- Мы благодарим Элли Волкхаузен (Ellie Volckhausen), дизайнера обложки.
- Крис Лопер из IntelligentEnglish.com сверстал и выпустил как печатный вариант книги, так и версии для EPUB и Kindle. Его инструментарий DocBook существенно упрощает нашу жизнь.
- Спасибо всем, кто предоставил нам на Facebook столько полезной информации об учебном курсе.

Остается лишь поблагодарить наших студентов. Жаль, что у нас не хватит места, чтобы поблагодарить каждого, кто высказал свое мнение или указал на недостаток в формирующемся учебном курсе. Мы постарались удовлетворить вашу любознательность и объяснить то, что было непонятно. Спасибо вам.

Изучение Android

Начинающему программисту Android предстоит основательно потрудиться. Изучение Android — все равно что жизнь в другой стране: даже если вы говорите на местном языке, на первых порах вы все равно не чувствуете себя дома. Такое впечатление, что все окружающие понимают что-то такое, чего вы еще не усвоили. И даже то, что уже известно, в новом контексте оказывается попросту неправильным.

У Android существует определенная культура. Носители этой культуры общаются на Java, но знать Java недостаточно. Чтобы понять Android, необходимо изучить много новых идей и приемов. Когда оказываешься в незнакомой местности, полезно иметь под рукой путеводитель.

Здесь на помощь приходим мы. Мы, сотрудники Big Nerd Ranch, считаем, что каждый программист Android должен:

- *писать* приложения для Android;
- *понимать*, что он пишет.

Этот учебник поможет вам в достижении обеих целей. Мы обучали сотни профессиональных программистов Android. Мы проведем вас по пути разработки нескольких приложений Android, описывая новые концепции и приемы по мере надобности. Если на пути нам встретятся какие-то трудности, если что-то покажется слишком сложным или нелогичным, мы постараемся объяснить, как возникло такое состояние дел.

Такой подход позволит вам сходу применить полученные сведения — вместо того чтобы накопить массу теоретических знаний, а потом разбираться, как же их использовать на практике. Перевернув последнюю страницу, вы будете обладать всем опытом, необходимым для дальнейшей работы в качестве Android-разработчика.

Предварительные условия

Чтобы использовать эту книгу, читатель должен быть знаком с языком Java, включая такие концепции, как классы и объекты, интерфейсы, слушатели, пакеты, внутренние классы, анонимные внутренние классы и обобщенные классы.

Без знания этих концепций вы почувствуете себя в джунглях, начиная со страницы 2. Лучше начните с вводного учебника по Java и вернитесь к этой книге после его прочтения. Сейчас имеется много превосходных книг для начинающих; подберите нужный вариант в зависимости от своего опыта программирования и стиля обучения.

Если вы хорошо разбираетесь в концепциях объектно-ориентированного программирования, но успели малость подзабыть Java — скорее всего, все будет нормально. Мы приводим краткие напоминания о некоторых специфических возможностях Java (таких, как интерфейсы и анонимные внутренние классы). Держите учебник по Java наготове на случай, если вам понадобится дополнительная информация во время чтения.

Как работать с книгой

Книга основана на материалах пятидневного учебного курса в Big Nerd Ranch. Соответственно предполагается, что вы будете читать ее с самого начала. Каждая глава базируется на предшествующем материале, и пропускать главы не рекомендуется. Таким образом, это не справочник. Мы старались помочь в преодолении начального барьера, чтобы вы могли извлечь максимум пользы из существующих справочников и сборников рецептов.

На наших занятиях студенты прорабатывают эти материалы, но в обучении также задействованы и другие факторы — специальное учебное помещение, хорошее питание и удобная доска, группа заинтересованных коллег и преподаватель, отвечающий на вопросы.

Желательно, чтобы ваша учебная среда была похожа на нашу. В частности, вам стоит хорошенько выспаться и найти спокойное место для работы. Следующие факторы тоже пригодятся:

- Создайте учебную группу с друзьями или коллегами.
- Выделяйте время, в которое вы будете заниматься исключительно чтением книги.
- Примите участие в работе форума книги на сайте *forums.bignerdranch.com*.
- Найдите специалиста по Android, который поможет вам в трудный момент.

Структура книги

В этой книге мы напишем восемь приложений для Android. Два приложения очень просты, и на их создание уходит всего одна глава. Другие приложения часто оказываются более сложными, а самое длинное приложение занимает тринадцать

глав. Все приложения спроектированы так, чтобы продемонстрировать важные концепции и приемы и дать опыт их практического применения.

GeoQuiz — в первом приложении мы исследуем основные принципы создания проектов Android, активности, макеты и явные интенты.

CriminalIntent — самое большое приложение в книге предназначено для хранения информации о проступках ваших коллег по офису. Вы научитесь использовать фрагменты, интерфейсы «главное-детализированное представление», списковые интерфейсы, меню, камеру, неявные интенты и многое другое.

HelloMoon — маленький алтарь программы «Аполлон». При построении этого приложения вы узнаете больше о фрагментах, воспроизведении мультимедийного контента, ресурсах и локализации.

NerdLauncher — нестандартный лаунчер раскроет тонкости работы системы интентов и задач.

RemoteControl — в этом «игрушечном» приложении вы научитесь пользоваться стилями, графическими объектами списков состояний и другими инструментами, предназначенными для создания привлекательных пользовательских интерфейсов.

PhotoGallery — клиент Flickr для загрузки и отображения фотографий из общедоступной базы Flickr. Приложение демонстрирует работу со службами, многопоточное программирование, обращения к веб-службам и т. д.

DragAndDraw — в этом простом графическом приложении рассматривается обработка событий касания и создание нестандартных представлений.

RunTracker — приложение позволяет отслеживать перемещения и отображать их на карте. Вы научитесь пользоваться сервисом геопозиционирования, базами данных SQLite, загрузчиками и картами.

Упражнения

Многие главы завершаются разделом с упражнениями. Это ваша возможность применить полученные знания, покопаться в документации и отработать навыки самостоятельного решения задач.

Мы настоятельно рекомендуем выполнять упражнения. Возможность сойти с проторенного пути и найти собственный путь закрепит учебный материал и придаст вам уверенности в работе над собственными проектами.

Если же вы окажетесь в тупике, вы всегда сможете обратиться за помощью на форум *forums.bignerdranch.com.*

А вы любознательны?

В конце многих глав также имеются разделы «Для любознательных». В них приводятся углубленные объяснения или дополнительная информация по темам, представленным в главе. Содержимое этих разделов не является абсолютно необходимым, но мы надеемся, что оно покажется вам интересным и полезным.

Стиль программирования

Существуют три ключевых момента, в которых наши решения отличаются от повсеместно встречающихся в сообществе Android.

Мы используем анонимные классы для слушателей. В основном это дело вкуса. На наш взгляд, код получается более стройным. Реализация метода слушателя размещается непосредственно там, где вы хотите ее видеть. В высокопроизводительных приложениях анонимные внутренние классы могут создать проблемы, но в большинстве случаев они работают нормально.

После знакомства с фрагментами в главе 7 мы используем их во всех пользовательских интерфейсах. Мы твердо убеждены в правильности этого выбора. Многие Android-разработчики продолжают писать код на базе активностей, но мы считаем эту практику сомнительной. Когда вы освоитесь с фрагментами, работать с ними несложно. Фрагменты имеют очевидные преимущества перед активностями, включая гибкость при построении и представлении пользовательских интерфейсов, так что дело того стоит.

Мы пишем приложения, совместимые с устройствами Gingerbread и Froyo. Платформа Android изменилась с появлением версий Ice Cream Sandwich и Jelly Bean, а скоро еще и Key Lime Pie. Однако истина заключается в том, что на половине устройств продолжают использоваться версии Froyo и Gingerbread. (Различия между версиями Android с «вкусными» названиями рассматриваются в главе 6.)

По этой причине мы намеренно проведем вас через все сложности, связанные с написанием приложений, совместимых с Froyo или по крайней мере Gingerbread. Если сразу начать с последней платформы, то учить, обучать и программировать для Android становится проще. Но мы хотели, чтобы вы умели разрабатывать приложения для реального мира, в котором телефоны на базе Gingerbread все еще составляют более 40 % устройств.

Необходимые инструменты

Пакет ADT Bundle

Прежде всего вам понадобится пакет ADT (Android Developer Tools) Bundle. Он включает в себя:

- Eclipse — интегрированная среда разработки для Android. Поскольку среда Eclipse также написана на Java, ее можно установить на PC, Mac или компьютер с системой Linux. Пользовательский интерфейс Eclipse следует «родным» стандартам пользовательского интерфейса вашей машины, поэтому вид экрана может несколько отличаться от снимков, приведенных в книге.
- Android Developer Tools — плагин для Eclipse. В книге используется ADT версии 21.1. Проследите за тем, чтобы у вас была установлена эта или более новая версия.
- Android SDK — последняя версия Android SDK.

- Инструменты и платформенные средства Android SDK — средства отладки и тестирования приложений.
- Образ системы для эмулятора Android — позволяет создавать и тестировать приложения на различных виртуальных устройствах.

Загрузка и установка

Пакет ADT Bundle доступен на сайте разработчиков Android в виде одного zip-файла.

1. Загрузите пакет по адресу *http://developer.android.com/sdk/index.html*.
2. Распакуйте zip-файл в тот каталог, в котором вы хотите установить Eclipse и другие инструменты.
3. В распакованных файлах найдите и откройте каталог eclipse и запустите Eclipse.

Если вы работаете в системе Windows и среда Eclipse не запускается, возможно, вам следует установить пакет Java Development Kit (JDK6), который можно загрузить на сайте *www.oracle.com*.

Если у вас все равно остаются проблемы, обратитесь по адресу *http://developer.android.com/sdk/index.html* за дополнительной информацией.

Загрузка старых версий SDK

Пакет ADT Bundle предоставляет SDK и образ эмулируемой системы для последней платформы. Однако для тестирования приложений в более ранних версиях Android вам могут понадобиться другие платформы.

Компоненты любой платформы можно получить при помощи Android SDK Manager. В Eclipse выполните команду Window ▸ Android SDK Manager.

Для каждой версии, начиная с Android 2.2 (Froyo), мы рекомендуем выбрать и установить:

- SDK Platform;
- системный образ для эмулятора;
- Google API.

Учтите, что загрузка этих компонентов может потребовать времени.

Android SDK Manager также используется для загрузки новейших выпусков Android — например, новой платформы или обновленных версий инструментов.

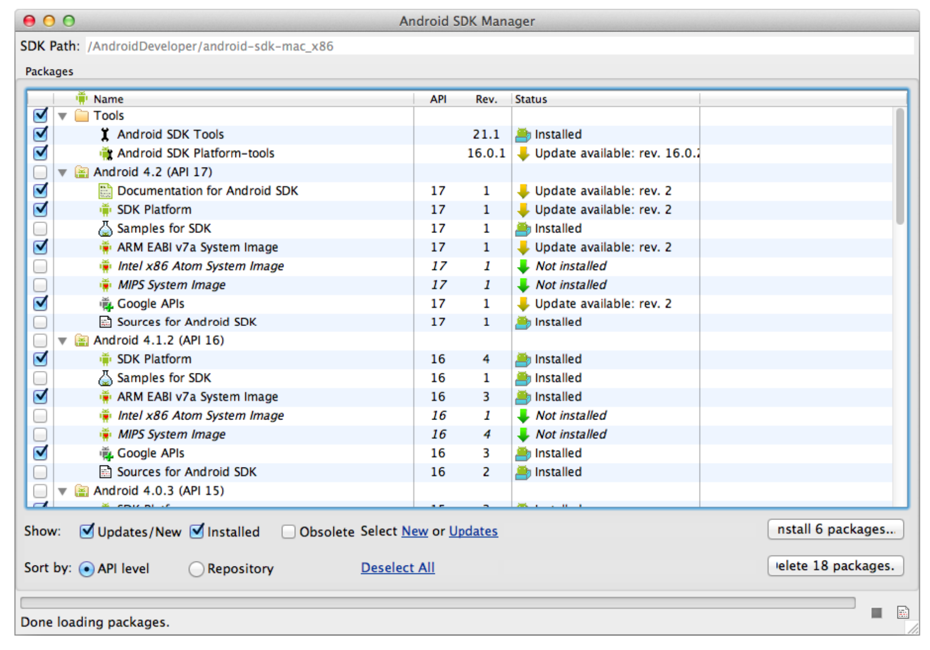


Рис. Android SDK Manager

Физическое устройство

Эмулятор удобен для тестирования приложений. Тем не менее также полезно иметь реальное устройство на базе Android для тестирования приложений. Скажем, последнее приложение в книге на эмуляторе работать не будет.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

1 Первое приложение Android

В первой главе представлено множество новых концепций и составляющих, необходимых для построения приложений Android. Не беспокойтесь, если к концу главы что-то останется непонятным — это нормально. Мы еще вернемся к этим концепциям в дальнейших главах и рассмотрим их более подробно.

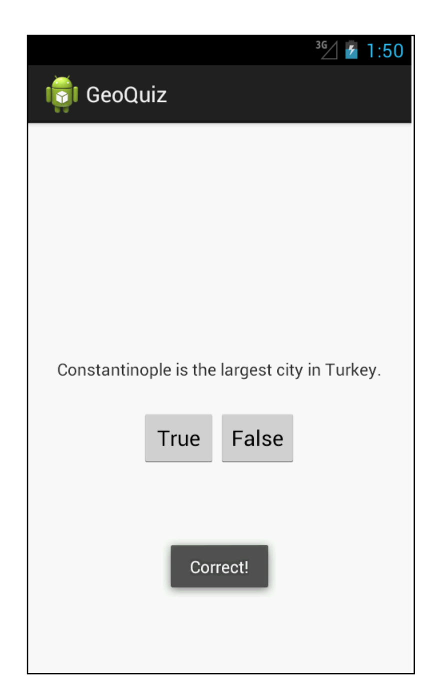


Рис. 1.1. Стамбул, а не Константинополь

Приложение, которое мы построим, называется GeoQuiz. Оно проверяет, насколько хорошо пользователь знает географию. Пользователь отвечает на вопрос, нажимая кнопку True или False, а GeoQuiz мгновенно сообщает ему результат.

На рис. 1.1 показан результат нажатия кнопки False.

Основы построения приложения

Приложение GeoQuiz состоит из *активности* (activity) и *макета* (layout):

- Активность представлена экземпляром `Activity` — класса из Android SDK. Она отвечает за взаимодействие пользователя с информацией на экране.

Чтобы реализовать функциональность, необходимую вашему приложению, разработчик

пишет субклассы `Activity`. В простом приложении бывает достаточно одного субкласса; в сложном приложении может потребоваться несколько.

GeoQuiz — простое приложение, поэтому в нем используется всего один субкласс `Activity` с именем `QuizActivity`. Класс `QuizActivity` управляет пользовательским интерфейсом, изображенным на рис. 1.1.

- Макет определяет набор объектов пользовательского интерфейса и их расположение на экране. Макет формируется из определений, написанных на языке XML. Каждое определение используется для создания объекта, выводимого на экране (например, кнопки или текста).

  Приложение GeoQuiz включает файл макета с именем activity_quiz.xml. Разметка XML в этом файле определяет пользовательский интерфейс, изображенный на рис. 1.1.

Отношения между `QuizActivity` и activity_quiz.xml изображены на рис. 1.2.

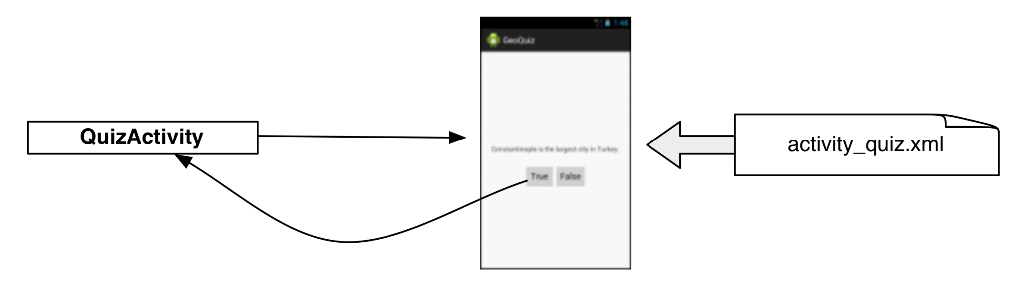


Рис. 1.2. QuizActivity управляет интерфейсом, определяемым в файле activity_quiz.xml

Учитывая все сказанное, давайте построим приложение.

Создание проекта Android

Работа начинается с создания *проекта Android*. Проект Android содержит файлы, из которых состоит приложение. Чтобы создать новый проект, откройте Eclipse и выполните команду File ▸ New ▸ Android Application Project, чтобы запустить мастера нового приложения.

В первом диалоговом окне введите имя приложения `GeoQuiz` (рис. 1.3). Имя проекта автоматически обновляется в соответствии с именем приложения. В поле Package Name введите имя пакета `com.bignerdranch.android.geoquiz`.

Обратите внимание: во введенном имени пакета используется схема «обратного DNS», согласно которой доменное имя вашей организации записывается в обратном порядке с присоединением суффиксов дополнительных идентификаторов. Эта схема обеспечивает уникальность имен пакетов и позволяет различать приложения на устройстве и в Google Play.

Следующие четыре поля настраивают ваше приложение для работы с разными версиями Android. Для приложения GeoQuiz достаточно настроек по умолчанию,

так что пока вы можете не обращать внимания на их содержимое. Разные версии Android более подробно рассматриваются в главе 6.

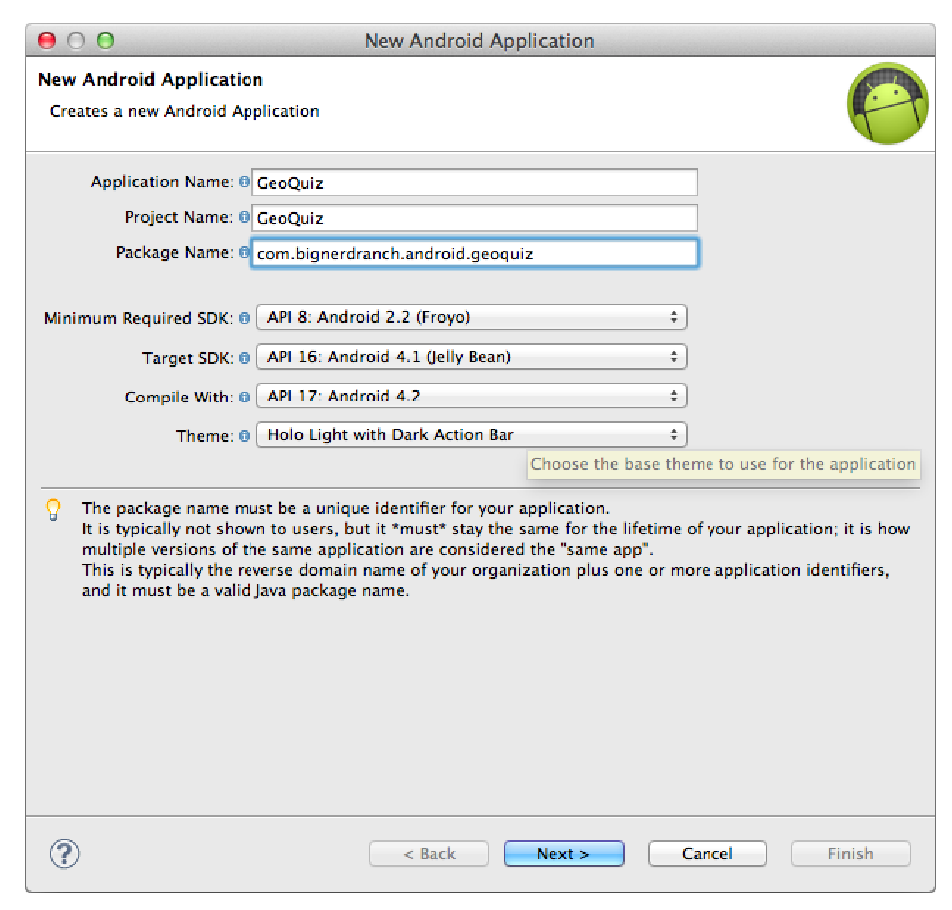


Рис. 1.3. Создание нового приложения

Инструментарий Android обновляется несколько раз в год, поэтому окно мастера на вашем компьютере может несколько отличаться от моего. Обычно это не создает проблем; принимаемые решения остаются практически неизменными. (Если ваше окно не имеет ничего общего с изображенным, значит, инструментарий изменился более радикально. Без паники. Загляните на форум книги *forums.bignerdranch.com*, и мы поможем вам найти обновленную версию.)

Щелкните на кнопке Next.

Во втором диалоговом окне снимите флажок Create custom launcher icon (рис. 1.4). Приложение GeoQuiz будет использовать значок запуска по умолчанию. Проследите за тем, чтобы флажок Create activity был установлен.

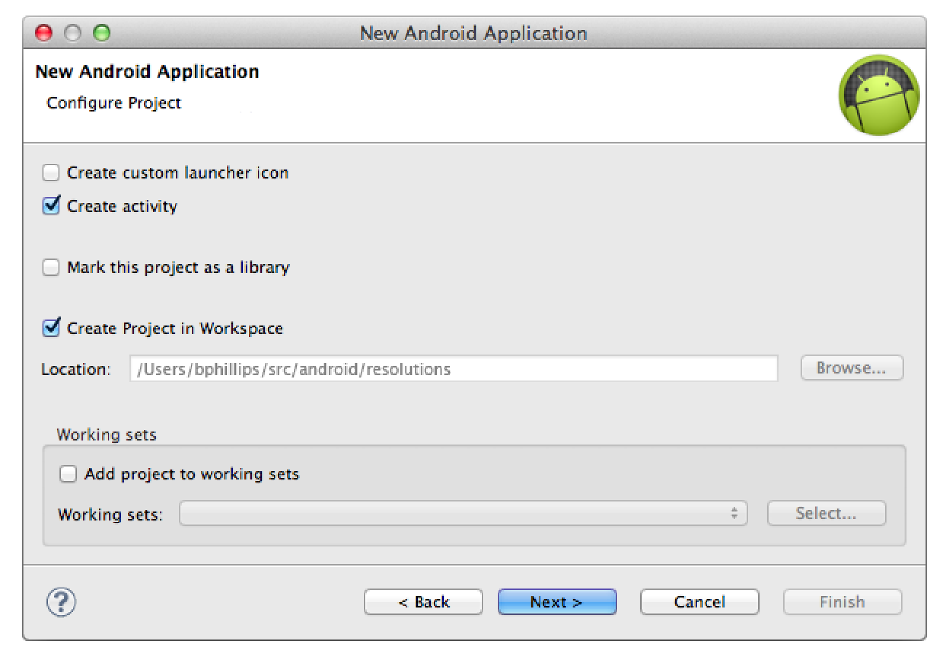


Рис. 1.4. Настройка параметров проекта

Щелкните на кнопке Next.

В следующем диалоговом окне (рис. 1.5) вам предлагается выбрать вид создаваемой активности. Выберите в списке строку Blank Activity (пустая активность).

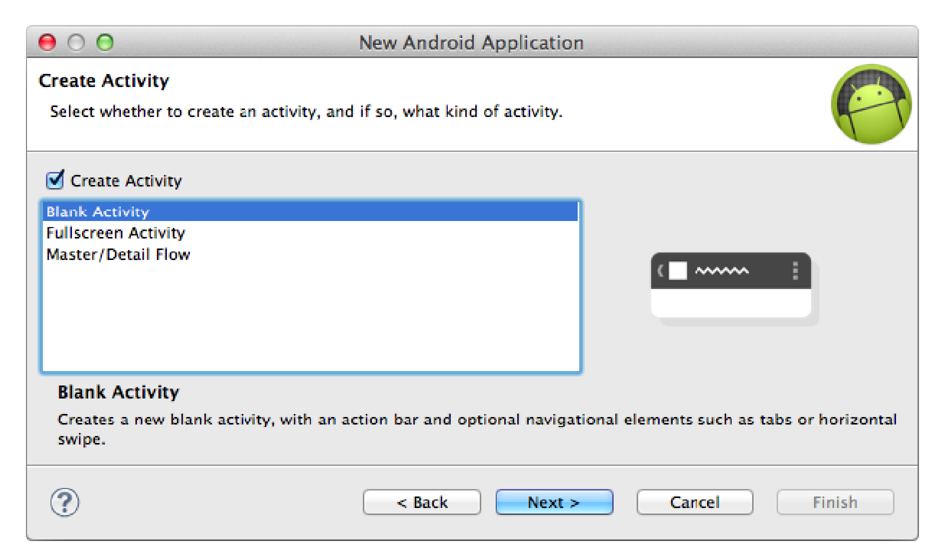


Рис. 1.5. Создание новой активности

Щелкните на кнопке Next.

В последнем диалоговом окне мастера введите имя субкласса активности `QuizActivity` (рис. 1.6). Обратите внимание на суффикс `Activity` в имени класса. Его присутствие не обязательно, но это очень полезное соглашение, которое стоит соблюдать.

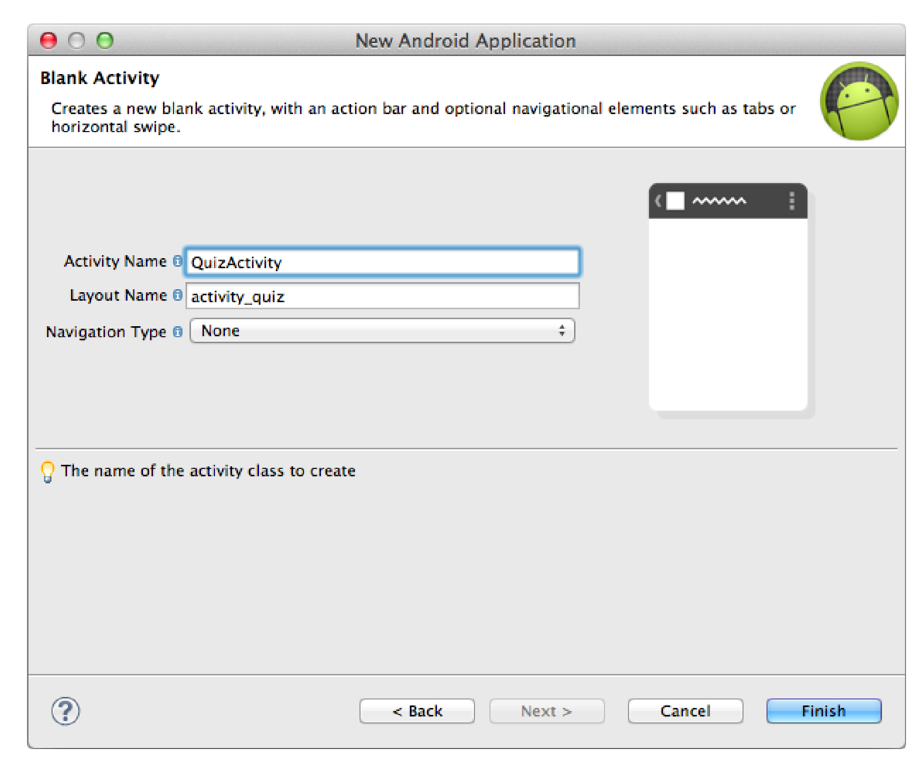


Рис. 1.6. Настройка новой активности

Имя макета автоматически заменяется на activity_quiz в соответствии с переименованием активности. Имя макета записывается в порядке, обратном имени активности; в нем используются символы нижнего регистра, а слова разделяются символами подчеркивания. Эту схему формирования имен рекомендуется применять как для макетов, так и для других ресурсов, о которых вы узнаете позднее.

Оставьте в списке Navigation Type значение None и щелкните на кнопке Finish. Среда Eclipse создает и открывает новый проект.

Навигация в Eclipse

Eclipse открывает ваш проект в *инструментальном окне* (workbench window), как показано на рис. 1.7. (Если вы только что установили Eclipse, закройте окно с приветствием Eclipse, чтобы открыть инструментальное окно.)

Слева находится панель Package Explorer. На ней выполняются операции с файлами, относящимися к вашему проекту.

В середине находится панель *редактора*. Чтобы вам было проще приступить к работе, Eclipse открывает в редакторе activity_quiz.xml.

В правой и нижней части инструментального окна также размещаются панели. Чтобы закрыть любую из панелей в правой части, щелкните на кнопке X рядом с именем панели (рис. 1.7). Панели в нижней части объединены в группу вкладок. Вместо того чтобы закрывать их, сверните всю группу при помощи кнопки в правом верхнем углу панели.

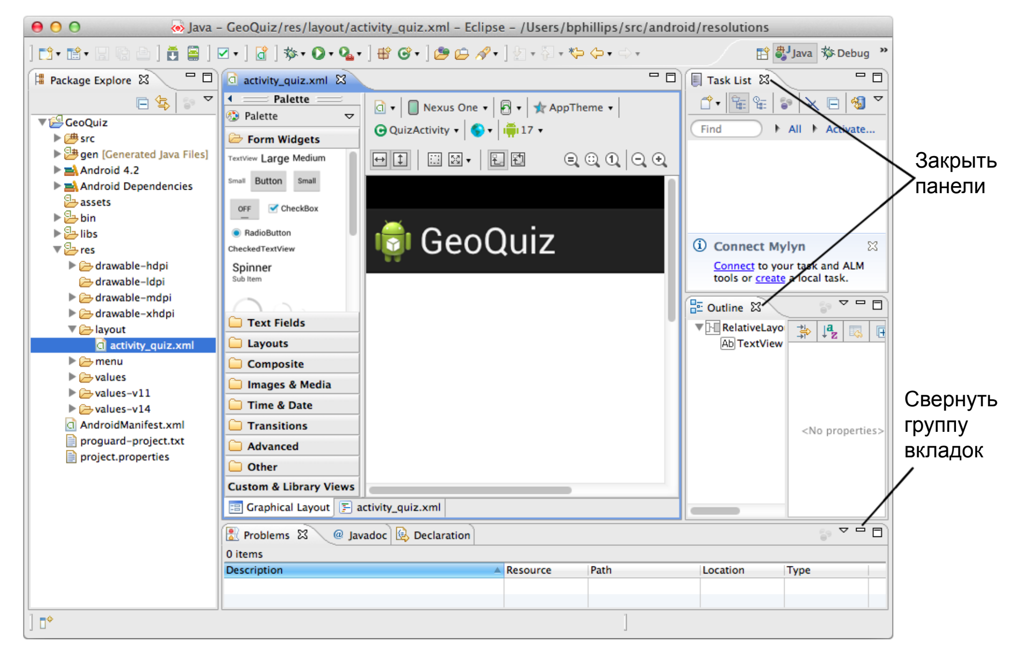


Рис. 1.7. Освобождение места в инструментальном окне

Когда вы сворачиваете панель, она закрепляется на полях инструментального окна Eclipse. Наведите указатель мыши на любой из маленьких значков на этих панелях инструментов, чтобы просмотреть имена свернутых панелей; щелкните кнопкой мыши, чтобы восстановить панель на экране.

Построение макета пользовательского интерфейса

По умолчанию Eclipse открывает activity_quiz.xml в конструкторе макетов Android, в котором отображается графическое представление макета. Этот режим бывает полезен, но сейчас мы будем работать в редакторе XML, чтобы вы лучше поняли, как работают макеты.

Чтобы перейти непосредственно к разметке XML, выберите вкладку activity_quiz.xml в нижней части редактора.

В настоящее время файл activity_quiz.xml определяет разметку для активности по умолчанию. Такая разметка часто изменяется, но XML будет выглядеть примерно так, как показано в листинге 1.1.

Листинг 1.1. Разметка для активности

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".QuizActivity" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />
</RelativeLayout>
```

Прежде всего обратите внимание на то, что файл activity_quiz.xml не начинается со строки с объявлением версии и кодировки:

```
<?xml version="1.0" encoding="utf-8"?>
```

В версии ADT 21 эта строка не является обязательной в файлах макетов Android. Впрочем, она все еще достаточно часто встречается в файлах.

Макет активности по умолчанию определяет два *виджета* (widgets): `RelativeLayout` и `TextView`.

Виджеты представляют собой структурные элементы, из которых составляется пользовательский интерфейс. Виджет может выводить текст или графику, взаимодействовать с пользователем или размещать другие виджеты на экране. Кнопки, текстовые поля, флажки — все это разновидности виджетов.

Android SDK включает множество виджетов, которые можно настраивать для получения нужного оформления и поведения. Каждый виджет является экземпляром класса `View` или одного из его субклассов (например, `TextView` или `Button`).

На рис. 1.8 показано, как выглядят на экране виджеты `RelativeLayout` и `TextView`, определенные в листинге 1.1.

Но это не виджеты, которые нам нужны. В интерфейсе QuizActivity задействованы пять виджетов:

- вертикальный виджет `LinearLayout`;
- `TextView`;
- горизонтальный виджет `LinearLayout`;
- две кнопки `Button`.

На рис. 1.9 показано, как из этих виджетов образуется интерфейс QuizActivity.

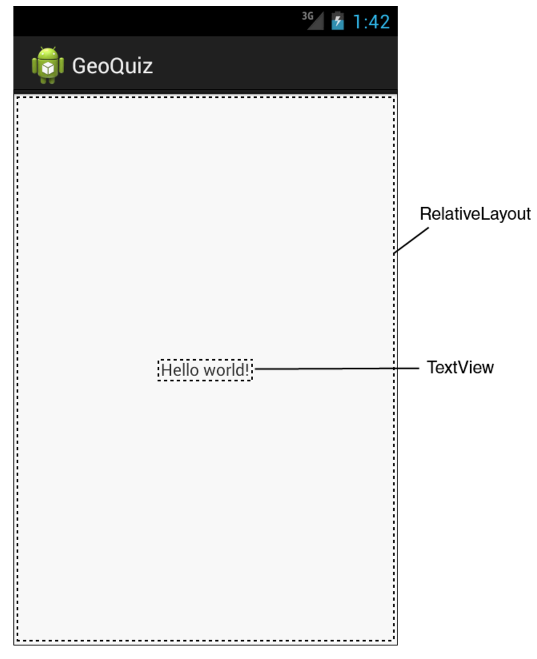


Рис. 1.8. Виджеты по умолчанию на экране

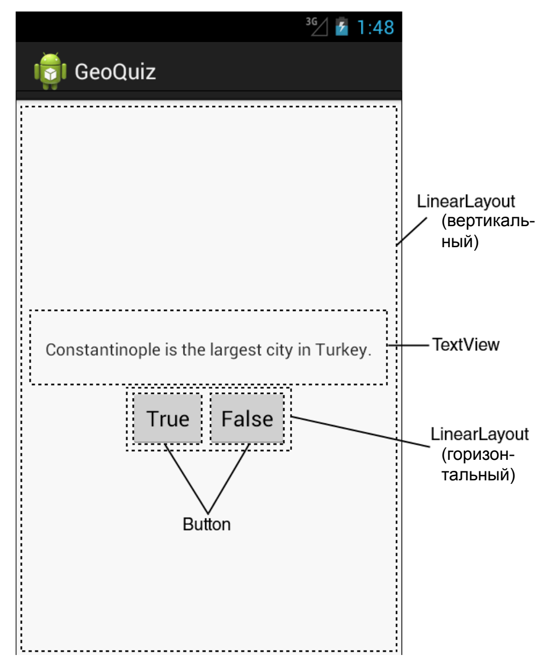


Рис. 1.9. Запланированное расположение виджетов на экране

Теперь нужно определить эти виджеты activity_quiz.xml.

Внесите в файл activity_quiz.xml изменения, представленные в листинге 1.2. Разметка XML, которую нужно удалить, выделена перечеркиванием, а добавляемая разметка XML выделена жирным шрифтом. Эти обозначения будут использоваться в книге.

Не беспокойтесь, если смысл вводимой разметки остается непонятным; скоро вы узнаете, как она работает. Но будьте внимательны: разметка макета не проверяется, и опечатки рано или поздно создадут проблемы.

В зависимости от вашей версии инструментария в трех строках, начинающихся с `android:text`, могут быть обнаружены ошибки. Пока не обращайте внимания. мы их скоро исправим.

Листинг 1.2. Определение виджетов в XML (activity_quiz.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".QuizActivity" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />
</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
      android:text="@string/false_button" />

  </LinearLayout>
</LinearLayout>
```

Сравните XML с пользовательским интерфейсом, изображенным на рис. 1.9. Каждому виджету в разметке соответствует элемент XML. Имя элемента определяет тип виджета.

Каждый элемент обладает набором *атрибутов* XML. Атрибуты можно рассматривать как инструкции по настройке виджетов.

Чтобы лучше понять, как работают элементы и атрибуты, полезно взглянуть на разметку с точки зрения иерархии.

Иерархия представлений

Виджеты входят в иерархию объектов `View`, называемую *иерархией представлений*. На рис. 1.10 изображена иерархия виджетов для разметки XML из листинга 1.2.

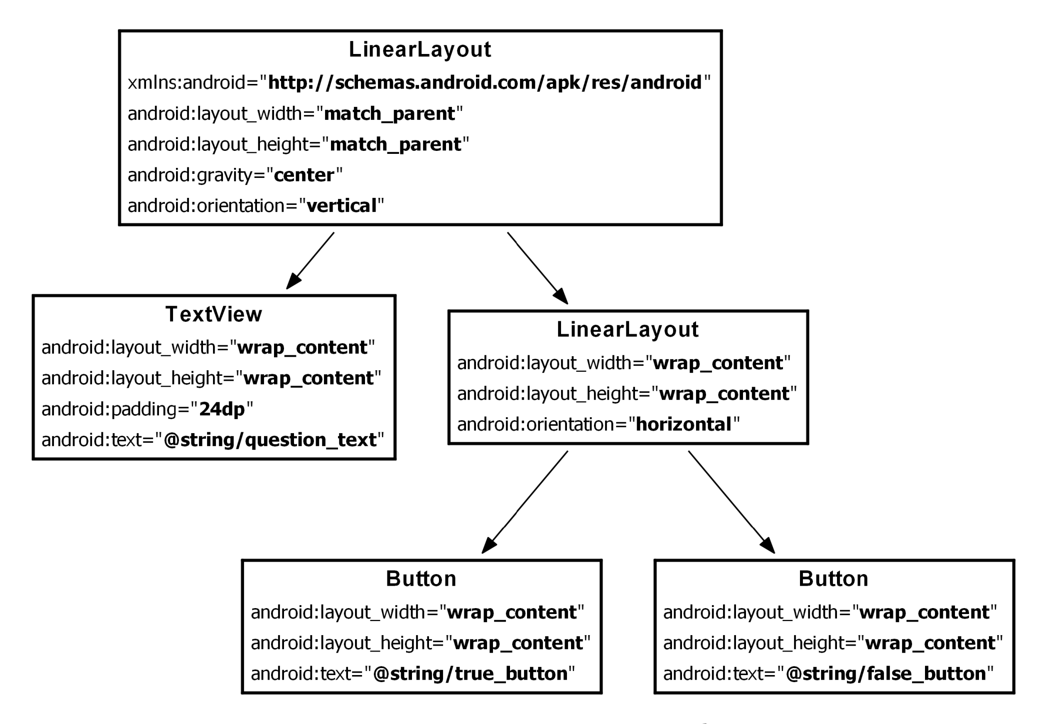


Рис. 1.10. Иерархия виджетов и атрибутов

Корневым элементом иерархии представлений в этом макете является элемент `LinearLayout`. В нем должно быть указано пространство имен XML ресурсов Android *http://schemas.android.com/apk/res/android*.

LinearLayout наследует от субкласса View с именем ViewGroup. Виджет ViewGroup предназначен для хранения и размещения других виджетов. Он используется в тех случаях, когда вы хотите выстроить виджеты в один столбец или строку. Другие субклассы ViewGroup — FrameLayout, TableLayout и RelativeLayout.

Если виджет содержится в ViewGroup, он называется *потомком* (child) ViewGroup. Корневой элемент LinearLayout имеет двух потомков: TextView и другой элемент LinearLayout. У LinearLayout имеются два собственных потомка Button.

Атрибуты виджетов

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

android:layout_width и android:layout_height

Атрибуты android:layout_width и android:layout_height, определяющие ширину и высоту, необходимы практически для всех разновидностей виджетов. Как правило, им задаются значения match_parent или wrap_content:

- match_parent — размеры представления определяются размерами родителя.
- wrap_content — размеры представления определяются размерами содержимого.

(Иногда в разметке встречается значение fill_parent. Это устаревшее значение эквивалентно match_parent.)

В корневом элементе LinearLayout атрибуты ширины и высоты равны match_parent. Элемент LinearLayout является корневым, но у него все равно есть родитель — представление, которое предоставляет Android для размещения иерархии представлений вашего приложения.

У других виджетов макета ширине и высоте задается значение wrap_content. На рис. 1.9 показано, как в этом случае определяются их размеры.

Виджет TextView чуть больше содержащегося в нем текста из-за атрибута android:padding="24dp". Этот атрибут приказывает виджету добавить заданный отступ вокруг содержимого при определении размера, чтобы текст вопроса не соприкасался с кнопкой. (Интересуетесь, что это за единицы — dp? Это пикселы, не зависящие от плотности (density-independent pixels), о которых будет рассказано в главе 8.)

android:orientation

Атрибут android:orientation двух виджетов LinearLayout определяет, как будут выстраиваться потомки — по вертикали или горизонтали. Корневой элемент LinearLayout имеет вертикальную ориентацию; у его потомка LinearLayout горизонтальная ориентация.

Порядок определения потомков определяет порядок их отображения на экране. В вертикальном элементе LinearLayout потомок, определенный первым, располагается выше остальных. В горизонтальном элементе LinearLayout первый потомок

является крайним левым. (Если только на устройстве не используется язык с письменностью справа налево — например, арабский или иврит; в этом случае первый потомок будет находиться в крайней правой позиции.)

android:text

Виджеты TextView и Button содержат атрибуты android:text. Этот атрибут сообщает виджету, какой текст должен отображаться

Обратите внимание: значения атрибутов представляют собой не строковые литералы, а ссылки на строковые ресурсы.

Строковый ресурс — строка, находящаяся в отдельном файле XML, который называется *строковым файлом*. Виджету можно назначить фиксированную строку (например, android:text="True"), но обычно так делать не стоит. Лучше размещать строки в отдельном файле, а затем ссылаться на них. В главе 15 вы увидите, как строковые ресурсы упрощают локализацию.

Строковые ресурсы, на которые мы ссылаемся в activity_quiz.xml, еще не существуют. Давайте исправим этот недостаток.

Создание строковых ресурсов

Каждый проект включает строковый файл по умолчанию с именем strings.xml.

Найдите на панели Package Explorer каталог res/values, раскройте его и откройте файл strings.xml.

Графический интерфейс нас пока не интересует; выберите вкладку strings.xml в нижней части редактора.

В шаблон уже включено несколько строковых ресурсов. Удалите неиспользуемую строку с именем hello_world и добавьте три новые строки для вашего макета.

Листинг 1.3. Добавление строковых ресурсов (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">GeoQuiz</string>
  <string name="hello_world">Hello, world!</string>
  <string name="question_text">Constantinople is the largest city in Turkey.</
string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="menu_settings">Settings</string>
</resources>
```

(Не удаляйте строку menu_settings — ваш проект содержит готовое меню. Удаление menu_settings вызовет каскадные ошибки в других файлах, относящихся к меню.)

Теперь по ссылке @string/false_button в любом файле XML проекта GeoQuiz вы будете получать строковый литерал "False" на стадии выполнения.

Сохраните файл strings.xml. Если в файле activity_quiz.xml оставались ошибки, связанные с отсутствием строковых ресурсов, они должны исчезнуть. (Если ошибки остались, проверьте оба файла — возможно, где-то допущена опечатка.)

Строковый файл по умолчанию называется strings.xml, но ему можно присвоить любое имя на ваше усмотрение. Проект может содержать несколько строковых файлов. Если файл находится в каталоге res/values/, содержит корневой элемент `resources` и дочерние элементы `string`, ваши строки будут найдены и правильно использованы приложением.

Предварительный просмотр макета

Макет готов и его можно просмотреть в графическом конструкторе. Прежде всего убедитесь в том, что файлы сохранены и не содержат ошибок. Затем вернитесь к файлу activity_quiz.xml и выберите вкладку Graphical Layout в нижней части редактора.

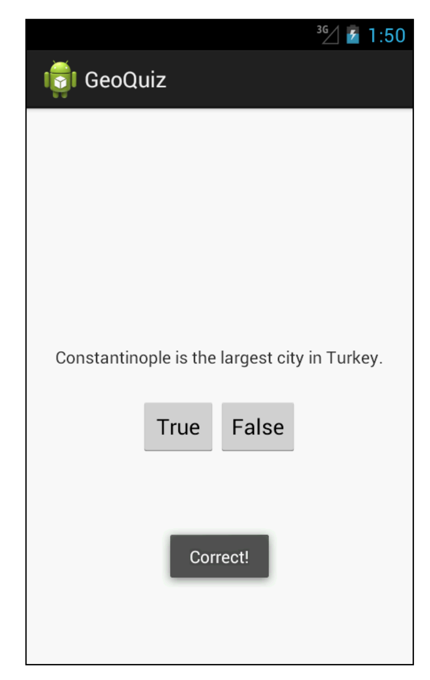


Рис. 1.11. Предварительный просмотр в графическом конструкторе макетов (activity_quiz.xml)

От разметки XML к объектам View

Как элементы XML в файле activity_quiz.xml превращаются в объекты `View`? Ответ на этот вопрос начинается с класса `QuizActivity`.

При создании проекта GeoQuiz был автоматически создан субкласс `Activity` с именем `QuizActivity`. Файл класса `QuizActivity` находится в каталоге src (в котором хранится Java-код вашего проекта).

На панели Package Explorer откройте каталог src, а затем содержимое пакета com.bignerdranch.android.geoquiz. Откройте файл QuizActivity.java и просмотрите его содержимое (листинг 1.4).

Листинг 1.4. Файл класса QuizActivity по умолчанию (QuizActivity.java)

```
package com.bignerdranch.android.geoquiz;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_quiz, menu);
        return true;
    }
}
```

(Если вы не видите все директивы `import`, щелкните на знаке ⊕ слева от первой директивы `import`, чтобы раскрыть список.)

Файл содержит два метода `Activity`: `onCreate(Bundle)` и `onCreateOptionsMenu(Menu)`.

Пока не обращайте внимания на метод `onCreateOptionsMenu(Menu)`. Мы вернемся к теме меню в главе 16.

Метод `onCreate(Bundle)` вызывается при создании экземпляра субкласса активности. Такому классу нужен пользовательский интерфейс, которым он будет управлять. Чтобы предоставить классу активности его пользовательский интерфейс, следует вызвать следующий метод `Activity`:

```
public void setContentView(int layoutResID)
```

Этот метод *заполняет* (inflates) макет и выводит его на экран. При заполнении макета создаются экземпляры всех виджетов в файле макета с параметрами, определяемыми его атрибутами. Чтобы указать, какой именно макет следует заполнить, вы передаете идентификатор ресурса макета.

Ресурсы и идентификаторы ресурсов

Макет представляет собой *ресурс*. Ресурсом называется часть приложения, которая не является кодом — графические файлы, аудиофайлы, файлы XML и т. д.

Ресурсы проекта находятся в подкаталоге каталога res. На панели Package Explorer видно, что файл activity_quiz.xml находится в каталоге res/layout/. Строковый файл, содержащий строковые ресурсы, находится в res/values/.

Для обращения к ресурсу в коде используется его идентификатор ресурса. Нашему макету назначен идентификатор ресурса `R.layout.activity_quiz`.

Чтобы просмотреть текущие идентификаторы ресурсов проекта GeoQuiz, откройте Package Explorer и раскройте содержимое каталога gen. Найдите и откройте файл R.java. Поскольку этот файл генерируется процессом сборки Android, вам не следует его изменять, о чем деликатно предупреждает надпись в начале файла.

Листинг 1.5. Текущие идентификаторы GeoQuiz

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
...
*/

package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070003;
    }
    public static final class layout {
        public static final int activity_quiz=0x7f030000;
    }
    public static final class menu {
       public static final int activity_quiz=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int false_button=0x7f040003;
        public static final int menu_settings=0x7f040006;
        public static final int question_text=0x7f040001;
        public static final int true_button=0x7f040002;
    }
    ...
}
```

Теперь понятно, откуда взялось имя `R.layout.activity_quiz` — это целочисленная константа с именем `activity_quiz` из внутреннего класса `layout` класса `R`.

Строкам также назначаются идентификаторы ресурсов. Мы еще не ссылались на строки в коде, но эти ссылки обычно выглядят так:

```
setTitle(R.string.app_name);
```

Android генерирует идентификатор ресурса для всего макета и для каждой строки, но не для отдельных виджетов из файла activity_quiz.xml. Не каждому виджету нужен

идентификатор ресурса. В этой главе мы будем взаимодействовать только с двумя кнопками, поэтому идентификаторы ресурсов нужны только им.

Чтобы сгенерировать идентификатор ресурса для виджета, включите в определение виджета атрибут `android:id`. В файле activity_quiz.xml добавьте атрибут `android:id` для каждой кнопки.

Листинг 1.6. Добавление идентификаторов кнопок (activity_quiz.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
      android:id="@+id/true_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:id="@+id/false_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />

  </LinearLayout>

</LinearLayout>
```

Обратите внимание: знак + присутствует в значениях `android:id`, но не в значениях `android:text`. Это связано с тем, что мы *создаем* идентификаторы, а на строки только *ссылаемся*.

Сохраните файл activity_quiz.xml. Вернитесь к файлу R.java и убедитесь в том, что во внутреннем классе `R.id` добавились два новых идентификатора ресурсов.

Листинг 1.7. Новые идентификаторы ресурсов (R.java)

```
public final class R {
    ...
    public static final class id {
        public static final int false_button=0x7f070001;
        public static final int menu_settings=0x7f070002;
        public static final int true_button=0x7f070000;
    }
...
```

Подключение виджетов к программе

Теперь, когда кнопкам назначены идентификаторы ресурсов, к ним можно обращаться в `QuizActivity`. Все начинается с добавления двух переменных.

Введите следующий код в QuizActivity.java. (Не используйте автозавершение; введите его самостоятельно.) После сохранения файла выводятся два сообщения об ошибках.

Листинг 1.8. Добавление полей (QuizActivity.java)

```
public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
...
}
```

Сейчас мы исправим ошибки, а пока обратите внимание на префикс `m` у имен двух полей (переменных экземпляров). Этот префикс соответствует схеме формирования имен Android, которая будет использоваться в этой книге.

Наведите указатель мыши на флаги ошибок слева от кода. Они сообщают об одинаковой проблеме: «`Button` не удается связать с типом» (Button cannot be resolved to a type).

Чтобы избавиться от ошибок, следует импортировать класс `android.widget.Button` в QuizActivity.java. Введите следующую директиву импортирования в начале файла:

```
import android.widget.Button;
```

А можно пойти по простому пути и провести организацию импорта.

Организация импорта

Команда «организации импорта» приказывает среде Eclipse проанализировать код и определить, что потребуется вашей программе из Java и Android SDK. Она импортирует все необходимое и удаляет ранее импортированные классы, которые не используются в программе.

Чтобы провести организацию импорта, нажмите:

- Command+Shift+O на Mac;
- Ctrl+Shift+O в Windows и Linux.

Ошибки должны исчезнуть. (Если они остались, поищите опечатки в коде и XML.)

Теперь вы можете подключить свои виджеты-кнопки. Процедура состоит из двух шагов:

- получение ссылок на заполненные объекты `View`;
- назначение для этих объектов слушателей, реагирующих на действия пользователя.

Получение ссылок на виджеты

В классе активности можно получить ссылку на заполненный виджет, для чего используется следующий метод `Activity`:

```
public View findViewById(int id)
```

Метод получает идентификатор ресурса виджета и возвращает объект `View`.

В файле QuizActivity.java по идентификаторам ресурсов ваших кнопок можно получить заполненные объекты и присвоить их полям. Учтите, что возвращенный объект `View` перед присваиванием необходимо преобразовать в `Button`.

Листинг 1.9. Получение ссылок на виджеты (QuizActivity.java)

```
public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button)findViewById(R.id.true_button);
        mFalseButton = (Button)findViewById(R.id.false_button);
    }
...
}
```

Назначение слушателей

Приложения Android обычно *управляются событиями* (event-driven). В отличие от программ командной строки или сценариев, такие приложения запускаются и ожидают наступления некоторого события — например, нажатия кнопки пользователем. (События также могут инициироваться ОС или другим приложением, но события, инициируемые пользователем, наиболее очевидны.)

Когда ваше приложение ожидает наступления конкретного события, мы говорим, что оно «прослушивает» данное событие. Объект, создаваемый для ответа на событие, называется *слушателем* (listener). Такой объект реализует *интерфейс слушателя* данного события.

Android SDK поставляется с интерфейсами слушателей для разных событий, поэтому вам не придется писать собственные реализации. В нашем случае прослушиваемым событием является «щелчок» на кнопке, поэтому слушатель должен реализовать интерфейс `View.OnClickListener`.

Начнем с кнопки True. В файле QuizActivity.java включите следующий фрагмент кода в метод onCreate(…) непосредственно после присваивания.

Листинг 1.10. Назначение слушателя для кнопки True (QuizActivity.java)

```
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mTrueButton = (Button)findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Пока ничего не делает, но скоро будет!
            }
        });

        mFalseButton = (Button)findViewById(R.id.false_button);
    }
}
```

(Если вы получите ошибку «View cannot be resolved to a type», проведите организацию импорта комбинацией Command+Shift+O или Ctrl+Shift+O для импортирования класса View.)

В листинге 1.10 назначается слушатель, информирующий о нажатии виджета Button с именем mTrueButton. Метод setOnClickListener(OnClickListener) получает в аргументе слушателя — а конкретнее объект, реализующий OnClickListener.

Анонимные внутренние классы

Слушатель реализован в виде анонимного внутреннего класса. Возможно, синтаксис не очевиден; просто запомните: все, что заключено во внешнюю пару круглых скобок, передается setOnClickListener(OnClickListener). В круглых скобках создается новый безымянный класс, вся реализация которого передается вызываемому методу.

```
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Пока ничего не делает, но скоро будет!
            }
        });
```

Все слушатели в этой книге будут реализованы в виде анонимных внутренних классов. В этом случае реализация методов слушателя находится непосредственно там, где вы хотите ее видеть, а мы избегаем затрат ресурсов на создание именованного класса, который будет использоваться только в одном месте.

Так как анонимный класс реализует OnClickListener, он должен реализовать единственный метод этого интерфейса onClick(View). Мы пока оставили реализацию

onClick(View) пустой, и компилятор не протестует. Интерфейс слушателя требует, чтобы метод onClick(View) был реализован, но не устанавливает никаких правил относительно того, *как именно* он будет реализован.

(Если ваши знания в области анонимных внутренних классов, слушателей или интерфейсов оставляют желать лучшего, полистайте учебник по Java перед тем, как продолжать, или хотя бы держите справочник под рукой.)

Назначьте аналогичного слушателя для кнопки False.

Листинг 1.11. Назначение слушателя для кнопки False (QuizActivity.java)

```
...
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Пока ничего не делает, но скоро будет!
        }
    });

    mFalseButton = (Button)findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Пока ничего не делает, но скоро будет!
        }
    });
}
```

Уведомления

Пора заставить кнопки делать что-то полезное. В нашем приложении каждая кнопка будет выводить на экран временное *уведомление* (toast) — короткое сообщение, которое содержит какую-либо информацию для пользователя, но не требует ни ввода, ни действий. Наши уведомления будут сообщать пользователю, правильно ли он ответил на вопрос (рис. 1.12).

Для начала вернитесь к файлу strings.xml и добавьте строковые ресурсы, которые будут отображаться в уведомлении.

Листинг 1.12. Добавление строк уведомлений (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">GeoQuiz</string>
  <string name="question_text">Constantinople is the largest city
in Turkey.</string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="correct_toast">Correct!</string>
  <string name="incorrect_toast">Incorrect!</string>
  <string name="menu_settings">Settings</string>
</resources>
```

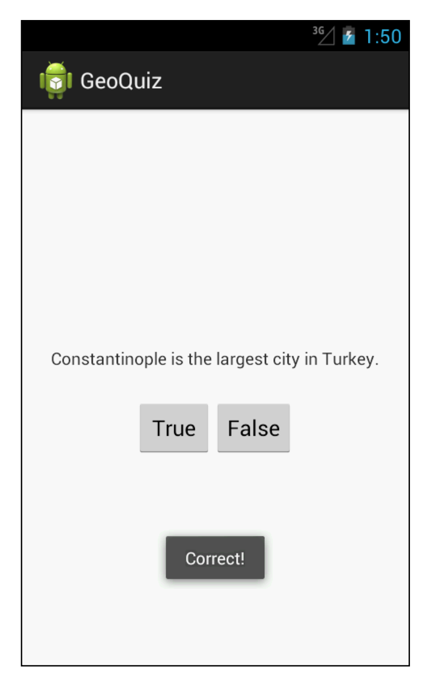


Рис. 1.12. Уведомление с информацией для пользователя

Уведомление создается вызовом следующего метода класса `Toast`:

```
public static Toast makeText(Context context, int resId, int duration)
```

Параметр `Context` обычно содержит экземпляр `Activity` (`Activity` является субклассом `Context`). Во втором параметре передается идентификатор ресурса строки, которая должна выводиться в уведомлении. Параметр `Context` необходим классу `Toast` для поиска и использования идентификатора ресурса строки. Третий параметр обычно содержит одну из двух констант `Toast`, определяющих продолжительность пребывания уведомления на экране.

После того как объект уведомления будет создан, вызовите `Toast.show()`, чтобы уведомление появилось на экране.

В классе `QuizActivity` вызов `makeText(…)` будет присутствовать в слушателе каждой кнопки (листинг 1.13). Вместо того чтобы вводить все вручную, попробуйте добавить эти вызовы с использованием функции автозавершения среды Eclipse.

Автозавершение

Автозавершение экономит много времени, так что с ним стоит познакомиться пораньше.

Начните вводить новый код из листинга 1.13. Когда вы доберетесь до точки после класса `Toast`, на экране появляется список методов и констант класса `Toast`.

Чтобы выбрать одну из рекомендаций, нажмите клавишу Tab, чтобы переключиться на подсказку автозавершения. (Если вы не собираетесь использовать автозавершение, просто продолжайте печатать. Без нажатия клавиши Tab или щелчка на окне подсказки подстановка не выполняется.)

Выберите в списке рекомендаций метод `makeText(Context, int, int)`. Механизм автозавершения добавляет вызов метода вместе с заполнителями аргументов.

Первый заполнитель выделяется автоматически; введите реальное значение — `QuizActivity.this`. Затем снова нажмите Tab, чтобы перейти к следующему заполнителю, и так далее, пока не будет введен весь код из листинга 1.13.

Листинг 1.13. Создание уведомлений (QuizActivity.java)

```
...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
    }
});
```

В вызове `makeText(…)` экземпляр `QuizActivity` передается в аргументе `Context`. Но как и следовало ожидать, просто передать `this` нельзя. В этом месте кода мы определяем анонимный класс, где `this` обозначает `View.OnClickListener`.

Благодаря использованию автозавершения вам не придется выполнять организацию импорта, чтобы класс `Toast` стал доступным. Когда вы соглашаетесь на рекомендацию автозавершения, необходимые классы импортируются автоматически.

Выполнение в эмуляторе

Для запуска приложений Android необходимо устройство — физическое или *виртуальное*. Виртуальные устройства работают под управлением эмулятора Android, включенного в поставку средств разработчика.

Чтобы создать виртуальное устройство Android (AVD, Android Virtual Device), выполните команду Window ▸ Android Virtual Device Manager. Когда на экране появится окно AVD Manager, щелкните на кнопке New... в правой части этого окна.

Открывается диалоговое окно с многочисленными параметрами настройки виртуального устройства. Выберите эмуляцию устройства Galaxy Nexus с Google APIs

(Google Inc.) – API Level 17, как показано на рис. 1.13. Если вы работаете в системе Windows, возможно, для правильной работы AVD вам придется уменьшить объем памяти (RAM) с 1024 до 512. Щелкните на кнопке OK.

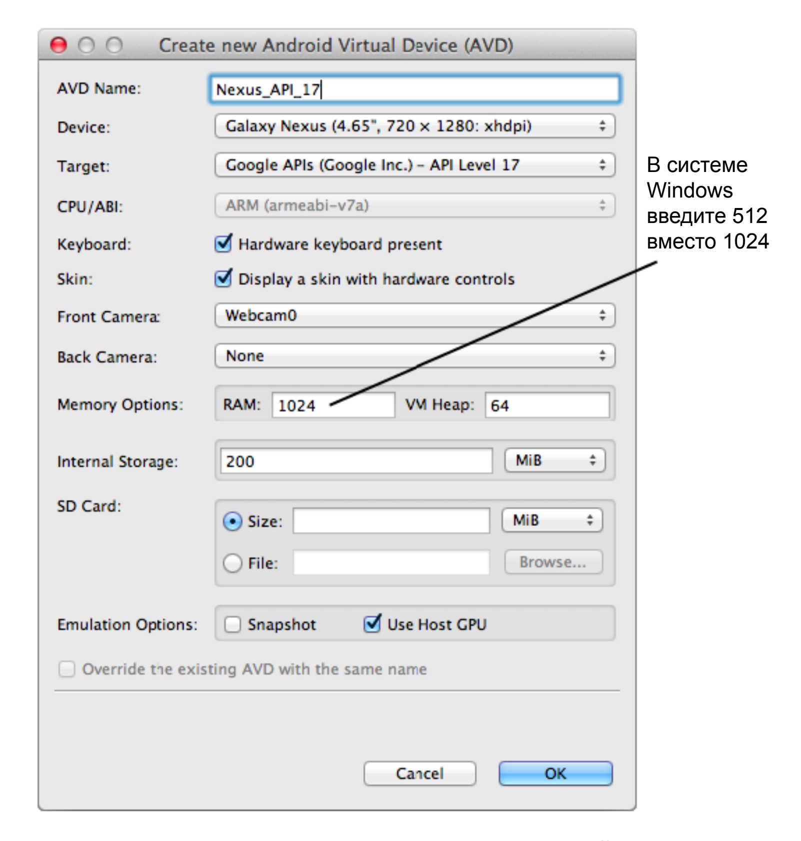


Рис. 1.13. Создание нового виртуального устройства Android

Когда виртуальное устройство будет создано, в нем можно запустить приложение GeoQuiz. На панели Package Explorer щелкните правой кнопкой мыши на папке проекта GeoQuiz. В контекстном меню выберите команду Run As ▸ Android Application. Eclipse находит созданное виртуальное устройство, устанавливает на нем пакет приложения и запускает приложение. Eclipse может спросить, хотите ли вы использовать автоматический мониторинг с LogCat — соглашайтесь.

Возможно, запуск эмулятора потребует некоторого времени, но вскоре приложение GeoQuiz запустится на созданном вами виртуальном устройстве. Понажимайте кнопки и оцените уведомления. (Если приложение запускается, когда вас нет поблизости, возможно, вам придется разблокировать AVD после возвращения. AVD работает как настоящее устройство и блокируется после некоторого бездействия.)

Если при запуске GeoQuiz или нажатии кнопки происходит сбой, в нижней части инструментального окна открывается панель LogCat. Ищите исключения в журнале; они будут выделяться красным цветом. В столбце Text указывается имя исключения и строка, в которой возникла проблема.

```
Text
at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21)
at android.app.Activity.performCreate(Activity.java:5008)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
```

Рис. 1.14. Исключение NullException в строке 21

Сравните свой код с кодом в книге и попытайтесь найти источник проблемы. Затем снова запустите приложение.

Не закрывайте эмулятор; не стоит ждать, пока он загружается, при каждом запуске. Вы можете остановить приложение кнопкой Back (U-образная стрелка), а затем снова запустить приложение из Eclipse, чтобы протестировать изменения.

Эмулятор полезен, но тестирование на реальном устройстве дает более точные результаты. В главе 2 мы запустим приложение GeoQuiz на физическом устройстве, а также расширим набор вопросов по географии.

Для любознательных: процесс построения приложений Android

Вероятно, у вас уже накопилось неотложных вопросов относительно того, как работает процесс построения приложений Android. Вы уже видели, что Eclipse строит проект автоматически в процессе его изменения, а не по команде. Во время построения инструментарий Android берет ваши ресурсы, код и файл AndroidManifest.xml (содержащий метаданные приложения) и преобразует их в файл .apk. Полученный файл подписывается отладочным ключом, что позволяет запускать его в эмуляторе.

(Чтобы распространять файл .apk среди пользователей, необходимо подписать его ключом публикации. Дополнительную информацию об этом процессе можно найти в документации разработчика Android по адресу *http://http://developer.android.com/tools/publishing/preparing.html*.)

Как содержимое activity_quiz.xml преобразуется в объекты `View` в приложении? В процессе построения утилита *aapt* (Android Asset Packaging Tool) компилирует ресурсы файлов макетов в более компактный формат. Откомпилированные ресурсы упаковываются в файл .apk. Затем, когда метод `setContentView(…)` вызывается в методе `onCreate(…)` класса `QuizActivity`, `QuizActivity` использует класс `LayoutInflater` для создания экземпляров всех объектов `View`, определенных в файле макета.

(Также классы представлений можно создать на программном уровне в классе активности вместо определения их в XML. Однако отделение презентационной логики от логики приложения имеет свои преимущества, главное из которых — использование изменений конфигурации, встроенное в SDK; мы подробнее поговорим об этом в главе 3.)

За дополнительной информацией о том, как работают различные атрибуты XML, и как происходит отображение представлений на экране, обращайтесь к главе 8.

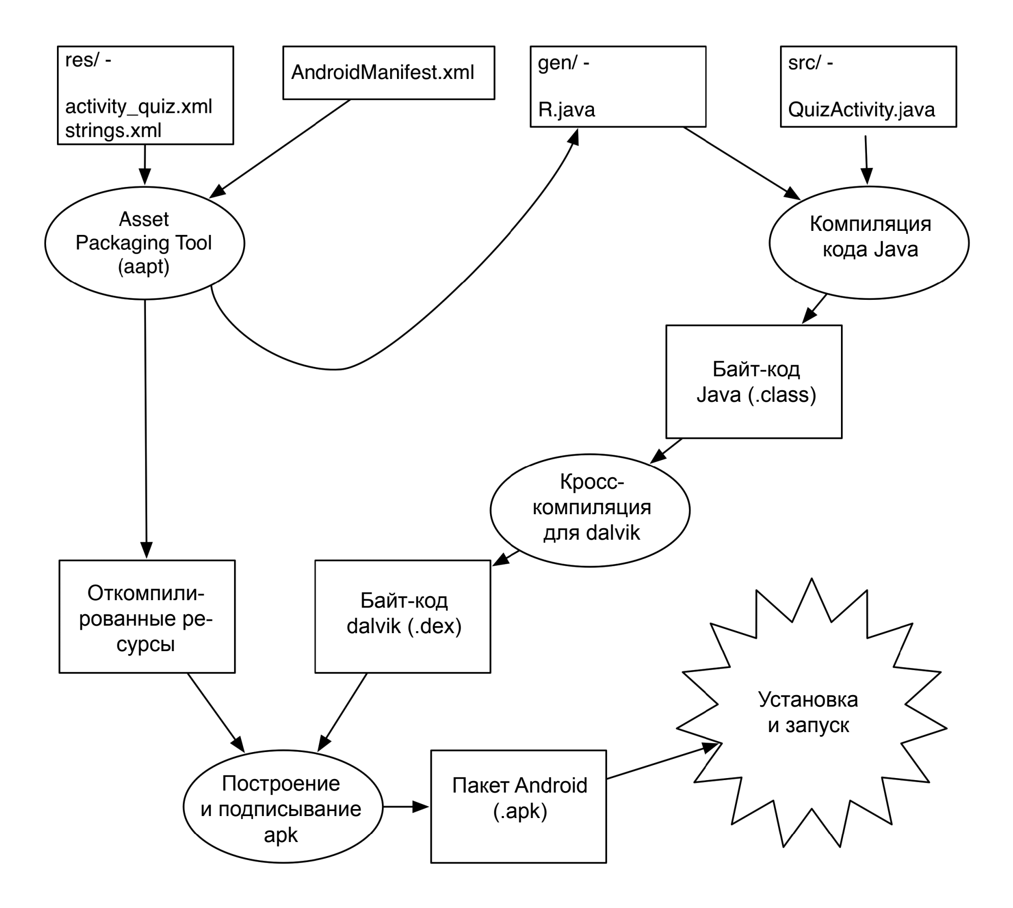


Рис. 1.15. Построение GeoQuiz

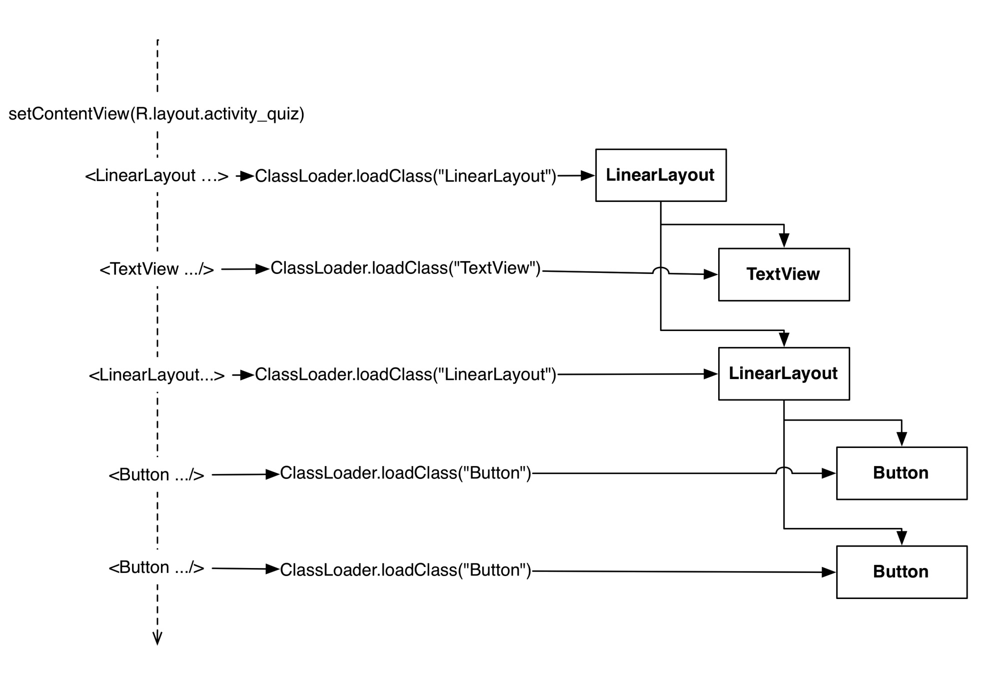


Рис. 1.16. Заполнение activity_quiz.xml

Средства построения программ Android

Все программы, которые мы запускали до настоящего времени, исполнялись из среды Eclipse. Этот процесс интегрирован в используемый плагин ADT — он вызывает стандартные средства построения программ Android (такие, как *aapt*), но сам процесс построения проходит под управлением Eclipse.

Может оказаться, что вам по каким-то своим причинам потребуется провести построение за пределами среды Eclipse. Для этого проще всего воспользоваться программой командной строки. Две самых популярных утилиты такого рода — *maven* и *ant*. *Ant* уступает по функциональности, но намного проще в использовании. Прежде всего выполните два шага:

- Убедитесь в том, что программа *ant* установлена и ее можно запустить.
- Убедитесь в том, что папки tools/ и platform-tools/ в Android SDK включены в пути поиска исполняемых файлов.

Теперь перейдите в каталог проекта и выполните следующую команду:

```
$ android update project -p .
```

Шаблон генератора проектов Eclipse не включает файл build.xml для *ant*. Первая команда генерирует файл build.xml за вас; вам остается лишь выполнить ее во второй раз.

Постройте проект. Чтобы построить и подписать отладочный файл .apk, выполните следующую команду в той же папке:

```
$ ant debug
```

Эта команда непосредственно строит программу. Она создает файл .apk, находящийся в папке bin/*имя-проекта*-debug.apk. Когда файл .apk будет создан, установите его следующей командой:

```
$ adb install bin/имя-проекта-debug.apk
```

Команда устанавливает приложение на подключенное устройство, но не запускает его — вы должны сделать это вручную.

2 Android и MVC

В этой главе мы обновим приложение GeoQuiz и включим в него дополнительные вопросы.

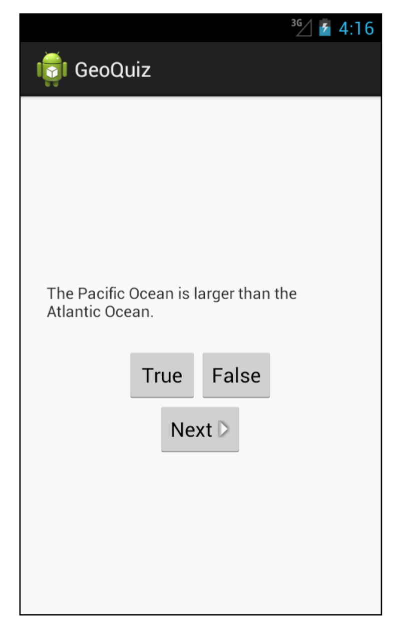


Рис. 2.1. Больше вопросов!

Для этого в проект GeoQuiz будет добавлен класс с именем `TrueFalse`. Экземпляр этого класса инкапсулирует один вопрос с ответом «да/нет».

Затем мы создадим массив объектов `TrueFalse`, с которым будет работать класс `QuizActivity`.

Создание нового класса

На панели Package Explorer щелкните правой кнопкой мыши на пакете com.bignerdranch.android.geoquiz и выберите команду New ▸ Class. Введите имя класса `TrueFalse`, оставьте суперкласс по умолчанию `java.lang.Object` и щелкните на кнопке Finish.

Рис. 2.2. Создание класса TrueFalse

Добавьте в файл TrueFalse.java два поля и конструктор:

Листинг 2.1. Добавление класса TrueFalse

```
public class TrueFalse {
    private int mQuestion;

    private boolean mTrueQuestion;

    public TrueFalse(int question, boolean trueQuestion) {
        mQuestion = question;
        mTrueQuestion = trueQuestion;
    }
}
```

Почему поле mQuestion объявлено с типом int, а не String? В нем будет храниться идентификатор ресурса (всегда int) строкового ресурса с текстом вопроса. Мы займемся созданием этих строковых ресурсов в следующем разделе. Переменная mTrueQuestion указывает, истинно или ложно утверждение в вопросе.

Для переменных необходимо определить get- и set-методы. Вводить их самостоятельно не нужно — проще приказать Eclipse сгенерировать реализации.

Генерирование get- и set-методов

Прежде всего следует настроить Eclipse на распознавание префикса m в полях классов и использование префикса is вместо get для логических переменных.

Откройте окно настроек Eclipse (меню Eclipse на Mac, команда Windows ▸ Preferences в системе Windows). В настройках Java выберите категорию Code Style.

В таблице Conventions for variable names: выберите строку Fields (рис. 2.3). Щелкните на кнопке Edit и введите префикс m для полей. Затем добавьте префикс s для статических полей. (В проекте GeoQuiz префикс s не используется, но он пригодится в будущих проектах.)

Проследите за тем, чтобы флажок Use 'is' prefix for getters that return boolean был установлен. Щелкните на кнопке OK.

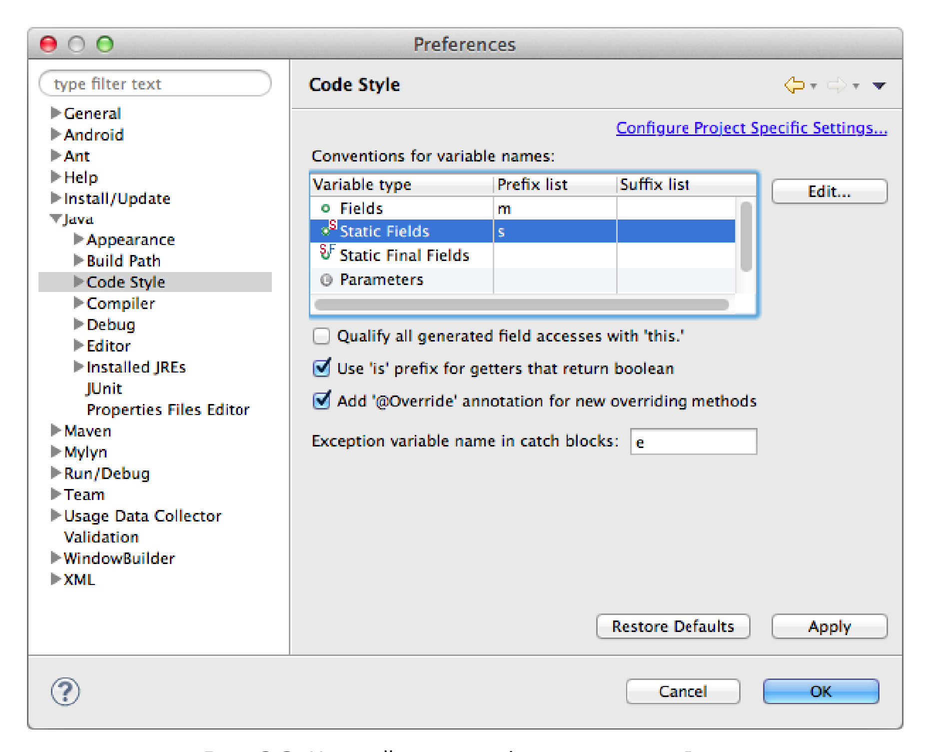


Рис. 2.3. Настройка стиля оформления кода Java

Зачем мы задавали эти префиксы? Если теперь приказать Eclipse сгенерировать `get`-метод для `mQuestion`, среда сгенерирует методы с именами `getQuestion()` вместо `getMQuestion()` и `isTrueQuestion()` вместо `isMTrueQuestion()`.

Вернитесь к файлу TrueFalse.java, щелкните правой кнопкой мыши после конструктора и выберите команду Source ▸ Generate Getters And Setters... Щелкните на кнопке Select All, чтобы сгенерировать `get`- и `set`-метод для каждой переменной.

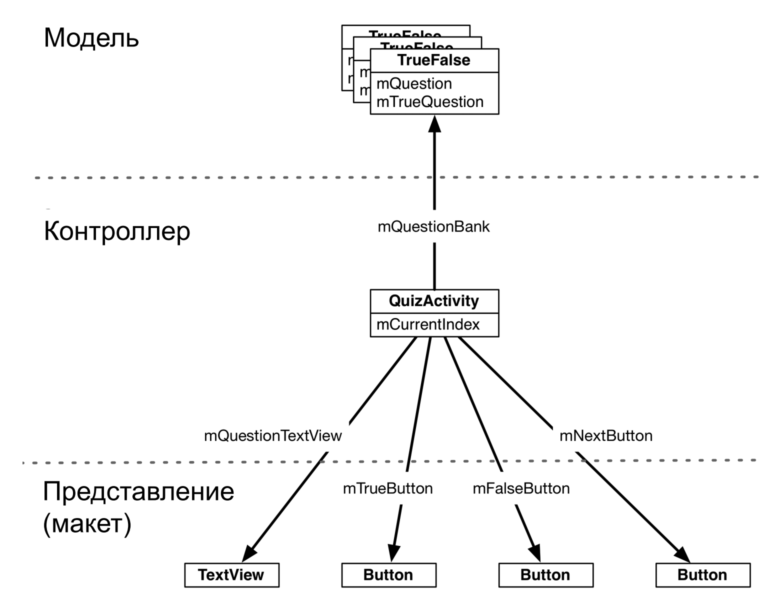


Рис. 2.4. Связи между объектами GeoQuiz

Щелкните на кнопке OK. Eclipse генерирует код четырех методов.

Листинг 2.2. Сгенерированные get- и set-методы

```
public class TrueFalse {
    private int mQuestion;

    private boolean mTrueQuestion;

    public TrueFalse(int question, boolean trueQuestion) {
        mQuestion = question;
        mTrueQuestion = trueQuestion;
    }

    public int getQuestion() {
        return mQuestion;
    }

    public void setQuestion(int question) {
        mQuestion = question;
    }
```

```
    public boolean isTrueQuestion() {
        return mTrueQuestion;
    }

    public void setTrueQuestion(boolean trueQuestion) {
        mTrueQuestion = trueQuestion;
    }
}
```

Класс `TrueFalse` готов. Вскоре мы внесем изменения в `QuizActivity` для работы с `TrueFalse`, но для начала посмотрим, как фрагменты GeoQuiz будут работать вместе.

Класс `QuizActivity` должен создать массив объектов `TrueFalse`. В процессе работы он взаимодействует с `TextView` и тремя виджетами `Button` для вывода вопросов и предоставления обратной связи на ответы пользователя.

Архитектура «Модель-Представление-Контроллер» и Android

Вероятно, вы заметили, что объекты на рис. 2.4 разделены на три области: «Модель», «Контроллер» и «Представление». Приложения Android строятся на базе архитектуры, называемой «Модель-Представление-Контроллер», или сокращенно MVC (Model-View-Controller). Согласно канонам MVC, каждый объект приложения должен быть объектом модели, объектом представления или объектом контроллера.

- *Объект модели* содержит данные приложения и «бизнес-логику». Классы модели обычно проектируются для моделирования сущностей, с которыми имеет дело приложение — пользователь, продукт в магазине, фотография на сервере, вопрос «да/нет» и т. д. Объекты модели ничего не знают о пользовательском интерфейсе; их единственной целью является хранение и управление данными.

  В приложениях Android классы моделей обычно создаются разработчиком для конкретной задачи. Все объекты модели в вашем приложении составляют его *уровень модели*.

  Уровень модели GeoQuiz состоит из класса `TrueFalse`.

- *Объекты представлений* умеют отображать себя на экране и реагировать на ввод пользователя — например, касания. Простейшее правило: если вы видите что-то на экране — значит, это представление.

  Android предоставляет широкий набор настраиваемых классов представлений. Разработчик также может создавать собственные классы представлений. Объекты представления в приложении образуют *уровень представления*.

  В GeoQuiz уровень представления состоит из виджетов, заполненных по содержимому файла activity_quiz.xml.

- *Объекты контроллеров* связывают объекты представления и модели; они содержат «логику приложения». Контроллеры реагируют на различные события,

инициируемые объектами представлений, и управляют потоками данных между объектами модели и уровнем представления.

В Android контроллер обычно представляется субклассом `Activity`, `Fragment` или `Service`. (Фрагменты рассматриваются в главе 7, а службы — в главе 29.)

Уровень контроллера GeoQuiz в настоящее время состоит только из класса `QuizActivity`.

На рис. 2.5 показана передача управления между объектами в ответ на пользовательское событие — такое, как нажатие кнопки. Обратите внимание: объекты модели и представлений не взаимодействуют друг с другом напрямую; в любом взаимодействии участвуют «посредники»-контроллеры, получающие сообщения от одних объектов и передающие инструкции другим.



Рис. 2.5. Взаимодействия MVC при получении ввода от пользователя

Преимущества MVC

Приложение может обрастать функциональностью до тех пор, пока не станет слишком сложным для понимания. Разделение кода на классы упрощает проектирование и понимание приложения в целом; разработчик мыслит в контексте классов, а не отдельных переменных и методов.

Аналогичным образом разделение классов на уровни модели, представления и контроллера упрощает проектирование и понимание приложения; вы можете мыслить в контексте уровней, а не отдельных классов.

GeoQuiz не является сложным приложением, но преимущества разделения уровней проявляются и здесь. Вскоре мы обновим уровень представления GeoQuiz и добавим в него кнопку Next. При этом нам совершенно не нужно помнить о только что созданном классе `TrueFalse`.

MVC также упрощает повторное использование классов. Класс с ограниченными обязанностями лучше подходит для повторного использования, чем класс, который пытается заниматься всем сразу.

Например, класс модели `TrueFalse` ничего не знает о виджетах, используемых для вывода вопроса «да/нет». Это упрощает использование `TrueFalse` в приложении для разных целей. Например, если потребуется вывести полный список всех вопросов, вы можете использовать тот же объект, который используется для вывода всего одного вопроса.

Обновление уровня представления

После теоретического знакомства с MVC пора переходить к практике — обновим уровень представления GeoQuiz и включим в него кнопку Next.

В Android объекты уровня представления обычно заполняются на основе разметки XML в файле макета. Весь макет GeoQuiz определяется в файле activity_quiz.xml. В него следует внести изменения, представленные на рис. 2.6. (Для экономии места на рисунке не показаны атрибуты виджетов, оставшихся без изменений.)

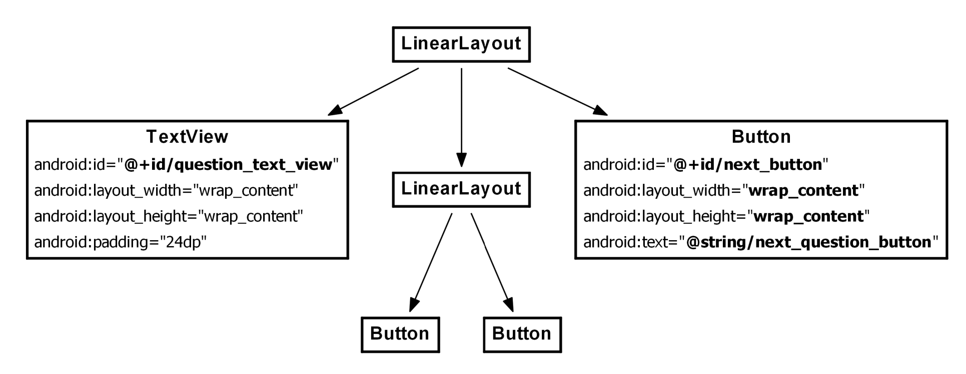


Рис. 2.6. Новая кнопка

Итак, на уровне представления необходимо внести следующие изменения:

- Удалите атрибут `android:text` из `TextView`. Жестко запрограммированный текст вопроса не должен присутствовать в определении.
- Назначьте `TextView` атрибут `android:id`. Идентификатор ресурса необходим виджету для того, чтобы мы могли задать его текст в коде `QuizActivity`.
- Добавьте новый виджет `Button` как потомка корневого элемента `LinearLayout`.

Вернитесь к файлу activity_quiz.xml и выполните все перечисленные действия.

Листинг 2.3. Новая кнопка и изменения в TextView (activity_quiz.xml)

```
<LinearLayout
  ... >

  <TextView
    android:id="@+id/question_text_view"
```

продолжение ⇗

Листинг 2.3 (*продолжение*)

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text"
    />

  <LinearLayout
    ... >

    ...
  </LinearLayout>

  <Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button" />

</LinearLayout>
```

Сохраните файл activity_quiz.xml. Возможно, на экране появится знакомая ошибка с информацией об отсутствующем строковом ресурсе.

Вернитесь к файлу res/values/strings.xml. Удалите строку вопроса и добавьте строку для новой кнопки.

Листинг 2.4. Обновленные строки (strings.xml)

```
...
  <string name="app_name">GeoQuiz</string>
  <string name="question_text">Constantinople is the largest city
                                in Turkey.</string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="next_button">Next</string>
  <string name="correct_toast">Correct!</string>
  ...
```

Пока файл strings.xml открыт, добавьте строки с вопросами по географии, которые будут предлагаться пользователю.

Листинг 2.5. Добавление строк вопросов (strings.xml)

```
...
  <string name="incorrect_toast">Incorrect!</string>
  <string name="menu_settings">Settings</string>
  <string name="question_oceans">The Pacific Ocean is larger than
    the Atlantic Ocean.</string>
  <string name="question_mideast">The Suez Canal connects the Red Sea
    and the Indian Ocean.</string>
  <string name="question_africa">The source of the Nile River is in Egypt.</string>
  <string name="question_americas">The Amazon River is the longest river
    in the Americas.</string>
  <string name="question_asia">Lake Baikal is the world\'s oldest and deepest
    freshwater lake.</string>
  ...
```

Обратите внимание на использование служебной последовательности \' для включения апострофа в последнюю строку. В строковых ресурсах могут использоваться все стандартные служебные последовательности, включая \n для обозначения новой строки.

Сохраните файлы. Вернитесь к файлу activity_quiz.xml и ознакомьтесь с изменениями макета в графическом конструкторе.

Пока это все, что относится к уровню представления GeoQuiz. Пора связать все воедино в классе контроллера QuizActivity.

Обновление уровня контроллера

В предыдущей главе в единственном контроллере GeoQuiz — QuizActivity — не происходило почти ничего. Он отображал макет, определенный в файле activity_quiz.xml, назначал слушателей для двух кнопок и организовывал выдачу уведомлений.

Теперь, когда у нас появились дополнительные вопросы, классу QuizActivity придется приложить дополнительные усилия для связывания уровней модели и представления GeoQuiz.

Откройте файл QuizActivity.java. Добавьте переменные для TextView и новой кнопки Button. Также создайте массив объектов TrueFalse и переменную для индекса массива.

Листинг 2.6. Добавление переменных и массива TrueFalse (QuizActivity.java)

```
public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;

    private TrueFalse[] mQuestionBank = new TrueFalse[] {
        new TrueFalse(R.string.question_oceans, true),
        new TrueFalse(R.string.question_mideast, false),
        new TrueFalse(R.string.question_africa, false)
        new TrueFalse(R.string.question_americas, true),
        new TrueFalse(R.string.question_asia, true),
    };

    private int mCurrentIndex = 0;
    ...
```

Программа несколько раз вызывает конструктор TrueFalse и создает массив объектов TrueFalse.

(В более сложном проекте этот массив создавался бы и хранился в другом месте. Позднее мы рассмотрим более правильные варианты хранения данных модели. А пока для простоты массив будет создаваться в контроллере.)

Мы собираемся использовать mQuestionBank, mCurrentIndex и методы доступа TrueFalse для вывода на экран серии вопросов.

Начнем с получения ссылки на TextView и задания тексту виджета вопроса с текущим индексом.

Листинг 2.7. Подключение виджета TextView (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        mTrueButton = (Button)findViewById(R.id.true_button);
        ...
    }
}
```

Сохраните файлы и проверьте возможные ошибки. Запустите программу GeoQuiz. Первый вопрос из массива должен отображаться в виджете TextView.

Теперь разберемся с кнопкой Next. Получите ссылку на кнопку, назначьте ей слушателя View.OnClickListener. Этот слушатель будет увеличивать индекс и обновлять текст TextView.

Листинг 2.8. Подключение новой кнопки (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
        ...
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.correct_toast,
                               Toast.LENGTH_SHORT).show();
            }
        });

        mNextButton = (Button)findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
```

```
                int question = mQuestionBank[mCurrentIndex].getQuestion();
                mQuestionTextView.setText(question);
            }
        });
    }
}
```

Обновление переменной `mQuestionTextView` осуществляется в двух разных местах. Лучше выделить этот код в закрытый метод, как показано в листинге 2.9. Далее остается лишь вызвать этот метод в слушателе `mNextButton` и в конце `onCreate(Bundle)` для исходного заполнения текста в представлении активности.

Листинг 2.9. Инкапсуляция в методе updateQuestion() (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getQuestion();
                mQuestionTextView.setText(question);
                updateQuestion();
            }
        });

        updateQuestion();
    }
}
```

Запустите GeoQuiz и протестируйте новую кнопку Next.

Итак, с вопросами мы разобрались — пора обратиться к ответам. И снова мы реализуем закрытый метод для инкапсуляции кода вместо того, чтобы вставлять одинаковый код в двух местах.

Сигнатура метода, который будет добавлен в `QuizActivity`, выглядит так:

```
private void checkAnswer(boolean userPressedTrue)
```

Метод получает логическую переменную, которая указывает, какую кнопку нажал пользователь: True или False. Ответ пользователя проверяется по ответу текущего объекта `TrueFalse`. Наконец, после определения правильности ответа метод создает уведомление для вывода соответствующего сообщения.

Включите в файл QuizActivity.java реализацию checkAnswer(boolean), приведенную в листинге 2.10.

Листинг 2.10. Добавление метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }

    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isTrueQuestion();

        int messageResId = 0;

        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

Включите в слушателя кнопки вызов checkAnswer(boolean), как показано в листинге 2.11.

Листинг 2.11. Вызов метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        mTrueButton = (Button)findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.incorrect_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(true);
            }
        });
```

```
        mFalseButton = (Button)findViewById(R.id.false_button);
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                               R.string.correct_toast,
                               Toast.LENGTH_SHORT).show();
                checkAnswer(false);
            }
        });

        mNextButton = (Button)findViewById(R.id.next_button);
        ...
    }
}
```

Программа GeoQuiz снова готова к работе. Давайте запустим ее на реальном устройстве.

Запуск на устройстве

В этом разделе мы займемся настройкой системы, устройства и приложения для выполнения GeoQuiz на физическом устройстве.

Подключение устройства

Прежде всего подключите устройство к системе. Если разработка ведется на Mac, ваша система должна немедленно распознать устройство. В системе Windows может потребоваться установка драйвера *adb* (Android Debug Bridge). Если Windows не может найти драйвер *adb*, загрузите его с сайта производителя устройства.

Чтобы убедиться в том, что устройство было успешно опознано, откройте панель Devices. Для этого проще всего открыть перспективу DDMS, щелкнув на кнопке DDMS в правом верхнем углу инструментального окна Eclipse. Панель Devices должна открыться в левой части инструментального окна. В списке должны присутствовать как AVD, так и физическое устройство.

Чтобы вернуться к редактору и другим панелям, щелкните на кнопке Java в правом верхнем углу инструментального окна.

Если у вас возникнут трудности с распознаванием устройства, прежде всего попробуйте заново инициализировать *adb*. На панели Devices щелкните на кнопке на стрелке, указывающей вниз, в правой верхней части панели. На экране появляется меню. Выберите в нем нижнюю команду Reset adb. Возможно, через непродолжительное время устройство появится в списке.

Если сброс не помогает, дополнительную информацию можно найти на сайте разработчиков Android. Начните со страницы *http://developer.android.com/tools/device.html* или обратитесь на форум книги *forums.bignerdranch.com* за дополнительной информацией о решении проблемы.

Настройка устройства для разработки

Чтобы вы могли тестировать приложения на своем устройстве, необходимо настроить его и разрешить установку приложений не из Google Play.

- На устройствах с Android 4.1 или ранее откройте меню Settings устройства и перейдите в раздел Applications. Убедитесь в том, что флажок Unknown sources установлен.
- На устройствах с Android 4.2 перейдите в раздел Settings ▸ Security и найдите флажок Unknown sources.

Также необходимо разрешить для устройства отладку USB.

- На устройствах с версиями Android до 4.0 откройте меню Settings ▸ Applications ▸ Development и найдите флажок USB debugging.
- На устройствах с Android 4.0 или 4.1 откройте меню Settings ▸ Developer options.
- В Android 4.2 флажок Developer options по умолчанию не отображается. Чтобы включить его, откройте меню Settings ▸ About Tablet/Phone и нажмите Build Number 7 раз. После этого вернитесь в меню Settings, найдите раздел Developer options и установите флажок USB debugging.

Как видите, настройки серьезно различаются между устройствами. Если у вас возникнут проблемы с включением отладки на вашем устройстве, обратитесь за помощью по адресу *http://developer.android.com/tools/device.html*.

Запустите GeoQuiz так, как это делалось ранее. Eclipse предлагает выбор между запуском на виртуальном устройстве и физическом устройстве, подключенном к системе. Выберите физическое устройство; GeoQuiz запускается на выбранном устройстве. (Если вам не предлагается выбрать устройство, а GeoQuiz запускается в эмуляторе, проверьте описанную ранее процедуру и убедитесь в том, что устройство подключено.)

Добавление значка

Приложение GeoQuiz работает, но пользовательский интерфейс смотрелся бы более привлекательно, если бы на кнопке Next была изображена стрелка, обращенная направо.

Изображение такой стрелки можно найти в файле решений этой книги (*http://www.bignerdranch.com/solutions/AndroidProgramming.zip*). Файл решений представляет собой набор проектов Eclipse — по одному для каждой главы.

Загрузите файл и откройте каталог 02_MVC/GeoQuiz/res. Найдите в нем подкаталоги drawable-hdpi, res/drawable-mdpi и drawable-xhdpi.

Суффиксы имен каталогов обозначают экранную плотность пикселов устройства.

mdpi	Средняя плотность (~160 dpi)
hdpi	Высокая плотность (~240 dpi)
xhdpi	Сверхвысокая плотность (~320 dpi)

(Также существует категория устройств низкой плотности `ldpi`, но устройства с такими экранами сейчас уже почти не встречаются.)

Каждый каталог содержит два графических файла — arrow_right.png и arrow_left.png. Эти файлы адаптированы для плотности пикселов, указанной в имени каталога.

В полноценных приложениях важно включить в проект графику для разных плотностей пикселов, поскольку это приводит к сокращению артефактов от масштабирования изображений. Вся графика в проекте устанавливается с приложением, а ОС выбирает наиболее подходящий вариант для конкретного устройства.

Добавление ресурсов в проект

Следующим шагом станет включение графических файлов в ресурсы приложения GeoQuiz.

На панели Package Explorer откройте содержимое каталога res и найдите три подкаталога, имена которых соответствуют именам из файла решений.

Скопируйте файлы из каждого каталога в соответствующий каталог Package Explorer.

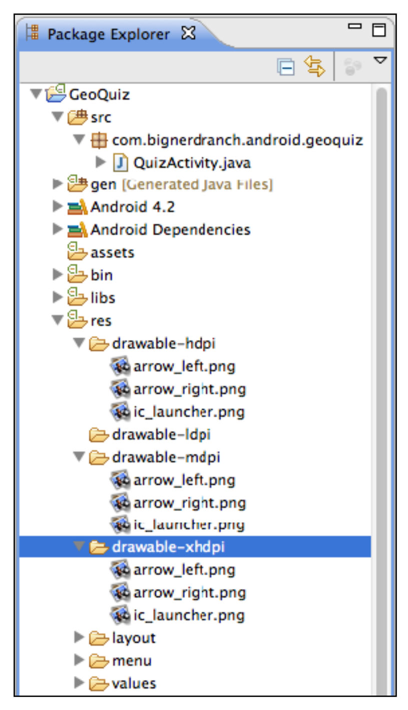


Рис. 2.7. Изображения стрелок в каталогах drawable проекта GeoQuiz

Если вы будете перетаскивать файлы мышью, на экране появляется окно, изображенное на рис. 2.8. Установите переключатель Copy Files, чтобы копировать файлы вместо создания ссылок на них.

Рис. 2.8. Копирование файлов (вместо создания ссылок)

Процесс включения графики в приложение чрезвычайно прост. Любому файлу .png, .jpg или .gif, добавленному в папку res/drawable, автоматически назначается идентификатор ресурса. (Учтите, что имена файлов должны быть записаны в нижнем регистре и не могут содержать пробелов.)

После того, как файлы будут скопированы, откройте файл gen/R.java и найдите новые идентификаторы ресурсов во внутреннем классе `R.drawable`. Таких идентификаторов всего два: `R.drawable.arrow_left` и `R.drawable.arrow_right`.

Эти идентификаторы ресурсов не уточняются плотностью пикселов, так что вам не нужно определять плотность пикселов экрана во время выполнения; просто используйте идентификатор ресурса в коде. При запуске приложения ОС определит изображение, подходящее для конкретного устройства.

Система ресурсов Android более подробно рассматривается в главе 3 и далее. А пока давайте заставим работать нашу стрелку.

Ссылки на ресурсы в XML

Для ссылок на ресурсы в коде используются идентификаторы ресурсов. Но мы хотим настроить кнопку Next так, чтобы в определении макета отображалась стрелка. Как включить ссылку на ресурс в разметку XML?

Да почти так же, только с немного измененным синтаксисом. Откройте файл activity_quiz.xml и добавьте два атрибута в определение виджета `Button`.

Листинг 2.12. Включение графики в кнопку Next (activity_quiz.xml)

```
<LinearLayout
  ... >
  ...

  <LinearLayout
    ... >
    ...
  </LinearLayout>

  <Button
    android:id="@+id/next_button"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_question_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
    />

</LinearLayout>
```

В ресурсах XML вы ссылаетесь на другой ресурс по его типу и имени. Ссылка на строку начинается с префикса `@string/`. Ссылка на графический ресурс начинается с префикса `@drawable/`.

Имена ресурсов и структура каталогов res более подробно рассматриваются, начиная с главы 3.

Запустите приложение GeoQuiz и насладитесь новым внешним видом кнопки. Протестируйте ее и убедитесь, что кнопка работает точно так же, как прежде.

Впрочем, в программе GeoQuiz скрывается ошибка. Во время выполнения GeoQuiz нажмите кнопку Next, чтобы перейти к следующему вопросу, а затем поверните устройство. (Если программа выполняется в эмуляторе, нажмите Control+F12/Ctrl+F12.)

После поворота мы снова видим первый вопрос. Почему это произошло и как исправить ошибку?

Ответы на эти вопросы связаны с жизненным циклом активности — этой теме посвящена глава 3.

Упражнения

Упражнения в конце главы предназначены для самостоятельной работы. Некоторые из них просты и рассчитаны на повторение того, что уже делалось в материале главы. Другие, более сложные упражнения потребуют логического мышления и навыков решения задач.

Трудно переоценить важность упражнений. Они закрепляют усвоенный материал, повышают уверенность в обретенных навыках и сокращают разрыв между изучением программирования Android и умением самостоятельно писать программы для Android.

Если у вас возникнут затруднения с каким-либо упражнением, сделайте перерыв, потом вернитесь и попробуйте еще раз. Если и эта попытка окажется безуспешной, обратитесь на форум книги `forums.bignerdranch.com`. Здесь вы сможете просмотреть вопросы и решения, полученные от других читателей, а также опубликовать свои вопросы и решения.

Чтобы избежать случайного повреждения текущего проекта, мы рекомендуем создать копию в Eclipse и практиковаться на ней.

Щелкните правой кнопкой мыши на проекте в Package Explorer, выберите команду Copy, затем снова щелкните правой кнопкой мыши и вставьте скопированный проект.

Вам будет предложено ввести имя нового проекта, который появляется на панели Package Explorer готовым к работе.

Упражнение. Добавление слушателя для TextView

Кнопка Next удобна, но было бы неплохо сделать так, чтобы пользователь мог перейти к следующему вопросу простым нажатием на виджете `TextView`.

Подсказка. Для `TextView` можно использовать слушателя `View.OnClickListener`, который использовался с `Button`, потому что класс `TextView` также является производным от `View`.

Упражнение. Добавление кнопки возврата

Добавьте кнопку для возвращения к предыдущему вопросу. Пользовательский интерфейс должен выглядеть примерно так, как показано на рис. 2.9.

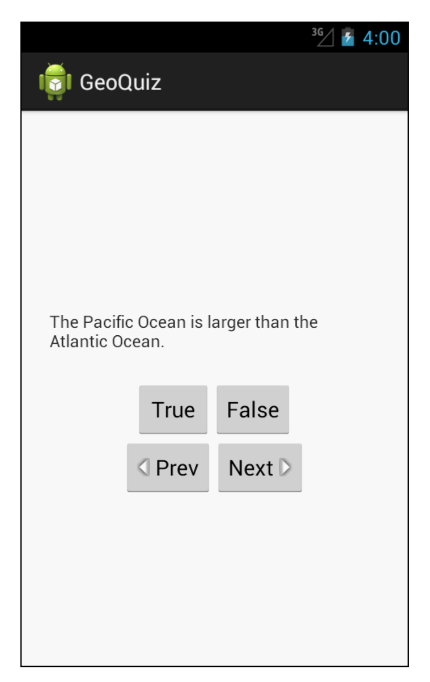


Рис. 2.9. Теперь с кнопкой возврата!

Это очень полезное упражнение. В нем вам придется вспомнить многое из того, о чем говорилось в двух последних главах.

Упражнение. От Button к ImageButton

Возможно, пользовательский интерфейс будет смотреться еще лучше, если на кнопках будут отображаться *только* значки.

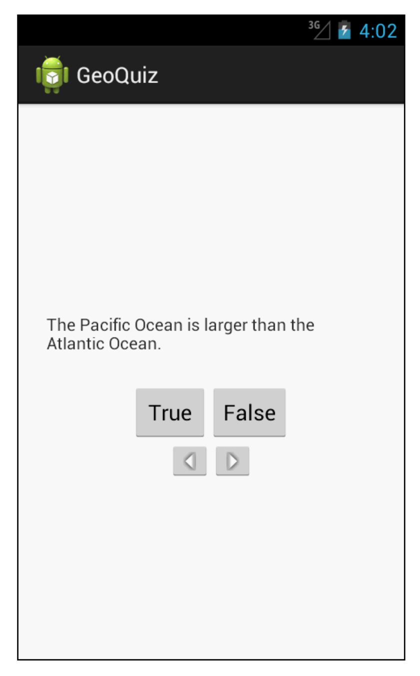


Рис. 2.10. Кнопки только со значками

Для этого оба виджета должны относиться к типу `ImageButton` (вместо обычного `Button`).

Виджет `ImageButton` является производным от `ImageView` — в отличие от виджета `Button`, производного от `TextView`.

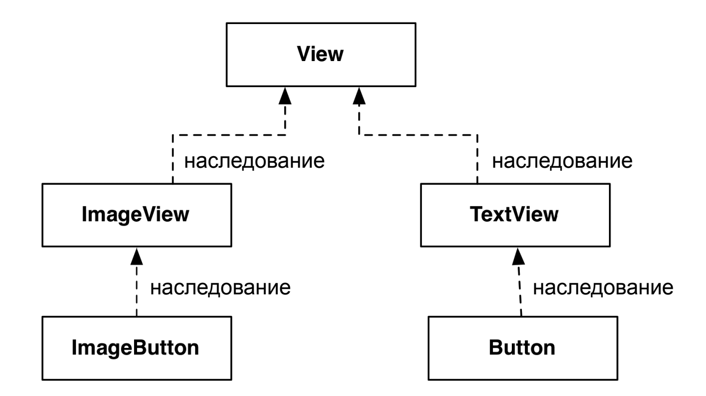


Рис. 2.11. Диаграмма наследования ImageButton и Button

Атрибуты `text` и `drawable` кнопки Next можно заменить одним атрибутом `ImageView`:

```
<~~Button~~ ImageButton
  android:id="@+id/next_button"
```

продолжение ⇗

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/next_question_button"
android:drawableRight="@drawable/arrow_right"
android:drawablePadding="4dp"
android:src="@drawable/arrow_right"
/>
```

Конечно, вам также придется внести изменения в QuizActivity, чтобы этот класс работал с ImageButton.

После того как вы замените кнопки кнопками ImageButton, Eclipse выдаст предупреждение об отсутствии атрибута android:contentDescription. Этот атрибут обеспечивает доступность контента для читателей с ослабленным зрением. Строка, заданная этому атрибуту, читается экранным диктором (при включении соответствующих настроек в системе пользователя).

Наконец, добавьте атрибут android:contentDescription в каждый элемент ImageButton.

3 Жизненный цикл Activity

У каждого экземпляра `Activity` имеется жизненный цикл. Во время этого жизненного цикла активность переходит между тремя возможными состояниями: выполнение, приостановка и остановка. Для каждого перехода у `Activity` существует метод, который оповещает активность об изменении состояния. На рис. 3.1 изображен жизненный цикл активности, состояния и методы.



Рис. 3.1. Диаграмма состояний Activity

Субклассы `Activity` могут использовать методы, представленные на рис. 3.1, для выполнения необходимых действий во время критических переходов жизненного цикла активности.

Один из таких методов вам уже знаком — это метод `onCreate(Bundle)`. ОС вызывает этот метод после создания экземпляра активности, но до его отображения на экране.

Как правило, активность переопределяет `onCreate(…)` для подготовки пользовательского интерфейса:

- заполнение виджетов и их вывод на экран (вызов `setContentView(int)`);
- получение ссылок на заполненные виджеты;
- назначение слушателей виджетам для обработки взаимодействия с пользователем;
- подключение к внешним данным модели.

Важно понимать, что вы никогда не вызываете `onCreate(…)` или другие методы жизненного цикла `Activity` в своих приложениях. Вы переопределяете их в субклассах активности, а Android вызывает их в нужный момент времени.

Регистрация событий жизненного цикла Activity

В этом разделе мы переопределим методы жизненного цикла, чтобы отслеживать основные переходы жизненного цикла QuizActivity. Реализации ограничиваются регистрацией в журнале сообщения о вызове метода.

Создание сообщений в журнале

В Android класс android.util.Log отправляет журнальные сообщения в общий журнал системного уровня. Класс Log предоставляет несколько методов регистрации сообщений. Следующий метод чаще всего встречается в этой книге:

```
public static int d(String tag, String msg)
```

Имя «d» означает «debug» (отладка) и относится к уровню регистрации сообщений. (Уровни Log более подробно рассматриваются в последнем разделе этой главы.) Первый параметр определяет источник сообщения, а второй — его содержимое.

Первая строка обычно содержит константу TAG, значением которой является имя класса. Это позволяет легко определить источник конкретного сообщения.

В файле QuizActivity.java добавьте константу TAG в QuizActivity.

Листинг 3.1. Добавление константы TAG (QuizActivity.java)

```
public class QuizActivity extends Activity {

private static final String TAG = "QuizActivity";
    ...
}
```

Включите в onCreate(…) вызов Log.d(…) для регистрации сообщения.

Листинг 3.2. Включение команды регистрации сообщения в OnCreate(…) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);

        ...
```

Если Eclipse выдаст ошибку, относящуюся к классу Log, нажмите Command+Shift+O (Ctrl+Shift+O) для проведения организации импорта. Eclipse предложит выбрать импортируемый класс; выберите android.util.Log.

Теперь переопределите еще пять методов в QuizActivity.

Листинг 3.3. Переопределение методов жизненного цикла (QuizActivity.java)

```
    } // Конец onCreate(Bundle)

 @Override
 public void onStart() {
 super.onStart();
 Log.d(TAG, "onStart() called");
 }

 @Override
 public void onPause() {
 super.onPause();
 Log.d(TAG, "onPause() called");
 }

 @Override
 public void onResume() {
 super.onResume();
 Log.d(TAG, "onResume() called");
 }

 @Override
 public void onStop() {
 super.onStop();
 Log.d(TAG, "onStop() called");
 }

 @Override
 public void onDestroy() {
 super.onDestroy();
 Log.d(TAG, "onDestroy() called");
 }

 }
```

Обратите внимание на вызовы реализаций суперкласса перед регистрацией сообщений. Эти вызовы являются обязательными, причем реализация суперкласса должна вызываться *до* выполнения каких-либо критических операций в `onCreate(…)`; в других методах порядок менее важен.

Возможно, вы заметили аннотацию `@Override`. Она приказывает компилятору проследить за тем, чтобы класс действительно содержал переопределяемый метод. Например, компилятор сможет предупредить вас об опечатке в имени метода:

```
public class QuizActivity extends Activity {

    @Override
    public void onCreat(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }

    ...
```

Класс `Activity` не содержит метод `onCreat(Bundle)`, поэтому компилятор выдает предупреждение. Это позволит вам исправить опечатку вместо того, чтобы случайно реализовать `QuizActivity.onCreat(Bundle)`.

Использование LogCat

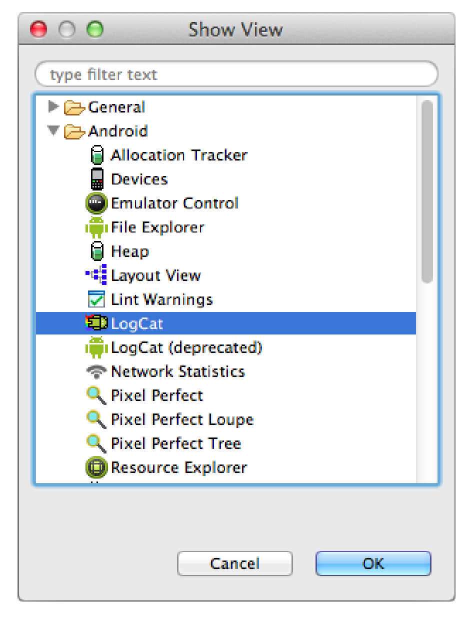


Рис. 3.2. Поиск программы LogCat

Чтобы просмотреть системный журнал во время работы приложения, используйте LogCat — программу, включенную в инструментарий Android SDK.

Чтобы запустить LogCat, выполните команду Window ▸ Show View ▸ Other... В папке Android найдите и выделите LogCat, затем щелкните на кнопке OK (рис. 3.2).

LogCat открывается в правой половине экрана, а панель редактора становится слишком мелкой. Было бы удобнее, если бы панель LogCat размещалась в нижней части инструментального окна.

Чтобы переместить панель, перетащите корешок панели LogCat на панель инструментов в правом нижнем углу инструментального окна (рис. 3.3).

Панель LogCat закрывается, а ее значок появляется на панели инструментов. Щелкните на значке; LogCat открывается в нижней части окна.

Инструментальное окно Eclipse должно выглядеть примерно так, как показано на рис. 3.4. Вы можете изменить размеры внутренних панелей LogCat, перетаскивая их границы мышью (это относится к любым панелям в инструментальном окне Eclipse).

Запустите GeoQuiz; на панели LogCat начнут быстро появляться сообщения. Большинство из них генерируется системой. Прокрутите список и найдите свои сообщения. В столбце TAG панели LogCat выводится константа TAG, определенная для `QuizActivity`.

(Если никакие сообщения не выводятся, возможно, LogCat отслеживает не то устройство. Выполните команду Window ▸ Show View ▸ Other... и откройте панель Devices. Выберите устройство, на котором работает программа, и вернитесь к LogCat.)

Чтобы упростить поиск сообщений, можно отфильтровать вывод по константе TAG. В LogCat щелкните на зеленом значке + в верхней части левой внутренней панели; эта кнопка создает фильтр сообщений. Введите имя фильтра QuizActivity и введите строку QuizActivity в поле by Log Tag: (рис. 3.5).

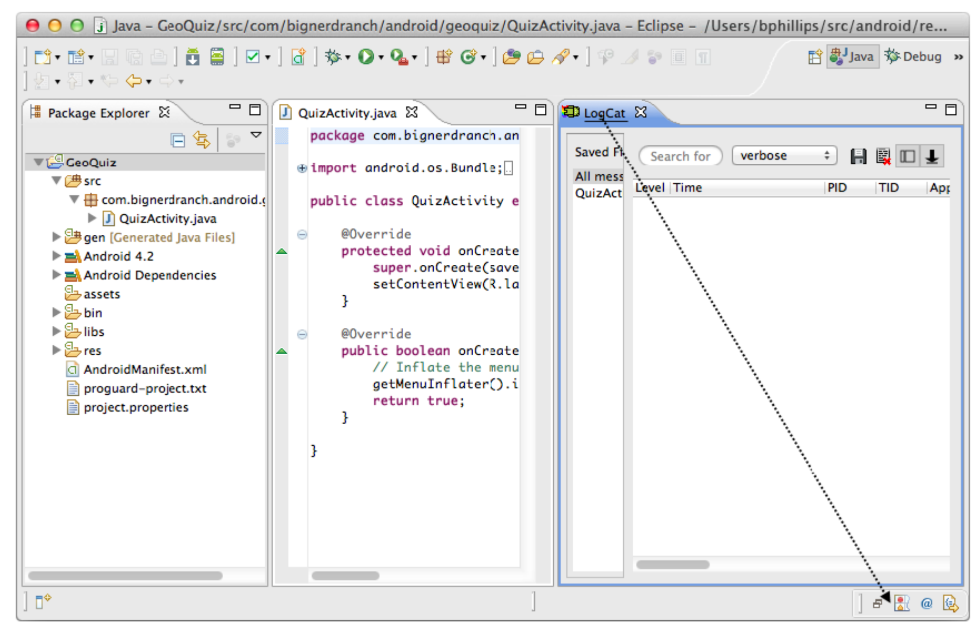


Рис. 3.3. Перетаскивание корешка LogCat на панель инструментов в правом нижнем углу

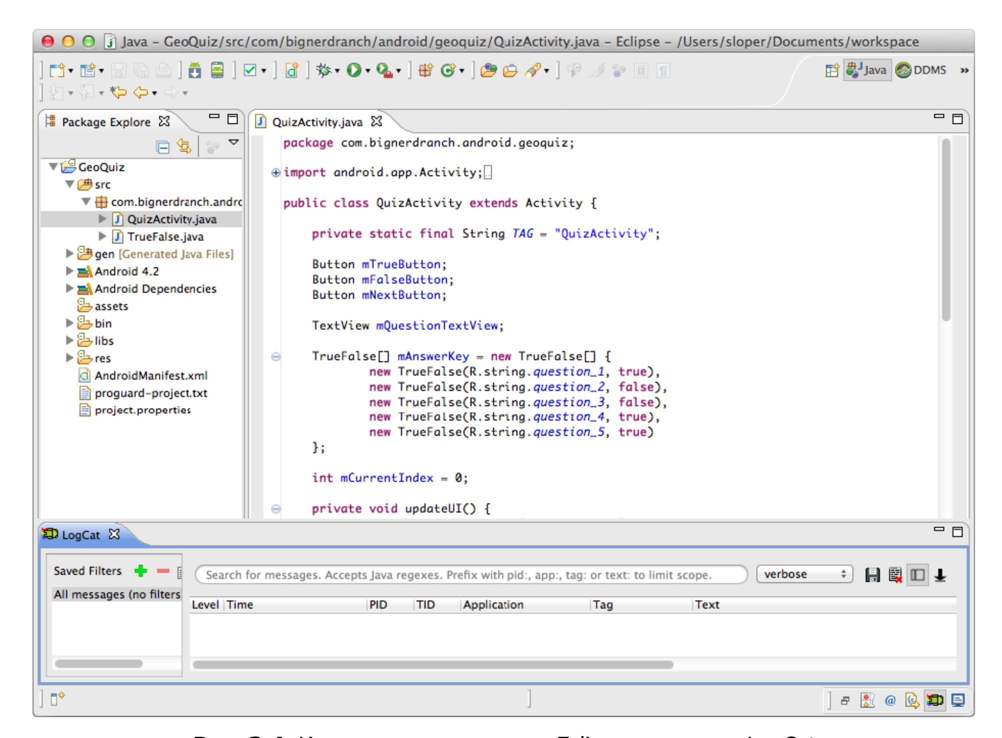


Рис. 3.4. Инструментальное окно Eclipse с панелью LogCat

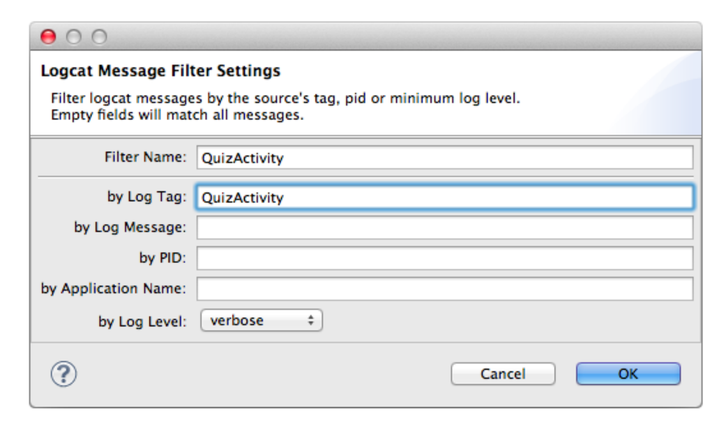


Рис. 3.5. Создание фильтра в LogCat

Щелкните на кнопке OK; открывается новая вкладка, в которой выводятся только сообщения с меткой `QuizActivity` (рис. 3.6). После запуска GeoQuiz были вызваны три метода жизненного цикла и был создан исходный экземпляр `QuizActivity`.

Level	Time	PID	TID	Application	Tag	Text
D	12-30 13:35:30.434	1097	1097	com.bignerdranch…	QuizActivity	onCreate
D	12-30 13:35:30.955	1097	1097	com.bignerdranch…	QuizActivity	onStart
D	12-30 13:35:31.054	1097	1097	com.bignerdranch…	QuizActivity	onResume

Рис. 3.6. Запуск GeoQuiz сопровождается созданием, запуском и *продолжением* активности

(Если вы не видите отфильтрованный список, выберите фильтр QuizActivity на левой панели LogCat.)

А теперь немного поэкспериментируем. Нажмите на устройстве кнопку Back, а затем проверьте вывод LogCat. Активность приложения получила вызовы `onPause()`, `onStop()` и `onDestroy()`.

Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:32:45.014	1097	1097	com.bignerdranch…	QuizActivity	onCreate
D	12-30 12:32:45.755	1097	1097	com.bignerdranch…	QuizActivity	onStart
D	12-30 12:32:45.785	1097	1097	com.bignerdranch…	QuizActivity	onResume
D	12-30 12:48:59.245	1097	1097	com.bignerdranch…	QuizActivity	onPause
D	12-30 12:49:01.284	1097	1097	com.bignerdranch…	QuizActivity	onStop
D	12-30 12:49:01.284	1097	1097	com.bignerdranch…	QuizActivity	onDestroy

Рис. 3.7. Нажатие кнопки Back приводит к уничтожению активности

Нажимая кнопку Back, вы сообщаете Android: «Я завершил работу с активностью, и она мне больше не нужна». Android уничтожает активность, чтобы не избежать неэффективного расходования ограниченных ресурсов устройства.

Перезапустите приложение GeoQuiz. Нажмите кнопку Home и проверьте вывод LogCat. Ваша активность получила вызовы `onPause()` и `onStop()`, но не вызов `onDestroy()`.

Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:49:01.284	1097	1097	com.bignerdranch…	QuizActivity	onStop
D	12-30 12:49:01.284	1097	1097	com.bignerdranch…	QuizActivity	onDestroy
D	12-30 12:50:01.087	1097	1097	com.bignerdranch…	QuizActivity	onCreate
D	12-30 12:50:01.715	1097	1097	com.bignerdranch…	QuizActivity	onStart
D	12-30 12:50:01.715	1097	1097	com.bignerdranch…	QuizActivity	onResume
D	12-30 12:50:47.075	1097	1097	com.bignerdranch…	QuizActivity	onPause
D	12-30 12:50:49.945	1097	1097	com.bignerdranch…	QuizActivity	onStop

Рис. 3.8. Нажатие кнопки Home останавливает активность

Вызовите на устройстве диспетчер задач. На новых устройствах для этого следует нажать кнопку Recents рядом с кнопкой Home (рис. 3.9). На устройствах без кнопки Recents выполните долгое нажатие кнопки Home.

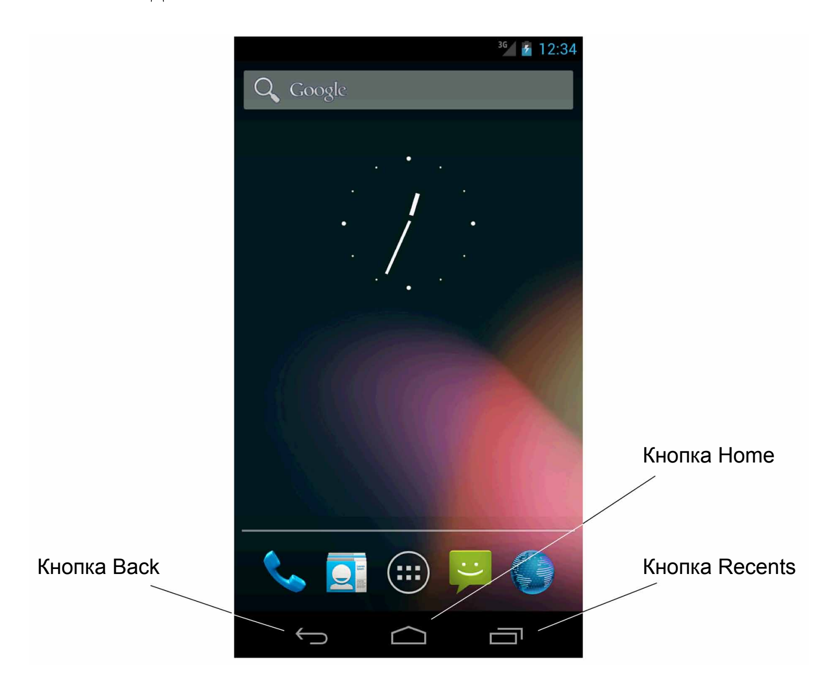


Рис. 3.9. Кнопки Home, Back и Recents

В диспетчере задач нажмите на приложении GeoQuiz и проверьте вывод LogCat. Активность запускается и продолжает работу, но создавать ее не нужно.

Нажатие кнопки Home сообщает Android: «Я сейчас займусь другим делом, но потом могу вернуться». Android приостанавливает активность, но старается не уничтожать ее на случай возвращения.

Тем не менее существование остановленной активности не гарантировано. Если системе потребуется занятая память, она уничтожает остановленные активности.

Наконец, представьте маленькое временное окно, которое только частично закрывает активность. При появлении такого окна находящаяся за ним активность приостанавливается и взаимодействие с ней невозможно. Выполнение активности будет продолжено, когда временное окно будет закрыто.

Далее в этой книге мы будем переопределять различные методы жизненного цикла активности для решения реальных задач. При этом применение каждого метода будет рассматриваться более подробно.

Повороты и жизненный цикл активности

А теперь вернемся к ошибке, обнаруженной в конце главы 2. Запустите GeoQuiz, нажмите кнопку Next для перехода к следующему вопросу, а затем поверните устройство. (Чтобы имитировать поворот в эмуляторе, нажмите Control+F12/Ctrl+F12).

После поворота GeoQuiz снова выводит первый вопрос. Чтобы понять, почему это произошло, просмотрите вывод LogCat.

LogCat

Saved Filters: All messages (no filters), QuizActivity

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope. | verbose

Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:54:07.914	1097	1097	com.bignerdranch…	QuizActivity	onCreate
D	12-30 12:54:08.575	1097	1097	com.bignerdranch…	QuizActivity	onStart
D	12-30 12:54:08.575	1097	1097	com.bignerdranch…	QuizActivity	onResume
D	12-30 13:00:31.635	1097	1097	com.bignerdranch…	QuizActivity	onPause
D	12-30 13:00:31.635	1097	1097	com.bignerdranch…	QuizActivity	onStop
D	12-30 13:00:31.635	1097	1097	com.bignerdranch…	QuizActivity	onDestroy
D	12-30 13:00:31.715	1097	1097	com.bignerdranch…	QuizActivity	onCreate
D	12-30 13:00:32.044	1097	1097	com.bignerdranch…	QuizActivity	onStart
D	12-30 13:00:32.134	1097	1097	com.bignerdranch…	QuizActivity	onResume

Рис. 3.10. QuizActivity умирает и возрождается

Когда вы поворачиваете устройство, экземпляр `QuizActivity`, который вы видели, уничтожается, и вместо него создается новый экземпляр. Снова поверните устройство — происходит еще один цикл уничтожения и возрождения.

Из-за этого и возникает ошибка. При каждом создании нового экземпляра `QuizActivity` переменная `mCurrentIndex` инициализируется 0, а пользователь начинает с первого вопроса. Вскоре мы исправим ошибку, но сначала повнимательнее разберемся, почему это происходит.

Конфигурации устройств и альтернативные ресурсы

Поворот приводит к изменению *конфигурации устройства*. Конфигурация устройства представляет собой набор характеристик, описывающих текущее состояние конкретного устройства. К числу характеристик, определяющих конфигурацию,

относится ориентация экрана, плотность пикселов, размер экрана, тип клавиатуры, режим стыковки, язык и многое другое.

Как правило, приложения предоставляют альтернативные ресурсы для разных конфигураций устройств. Пример такого рода нам уже встречался — вспомните, как мы включали в проект несколько изображений стрелки для разной плотности пикселов.

Плотность пикселов является фиксированным компонентом конфигурации устройства; она не может измениться во время выполнения. Напротив, некоторые компоненты (такие, как ориентация) *могут* изменяться при выполнении.

При изменении конфигурации во время выполнения может оказаться, что приложение содержит ресурсы, лучше подходящие для новой конфигурации. Чтобы увидеть, как работает этот механизм, мы создадим альтернативный ресурс, который Android найдет и использует при изменении ориентации экрана.

Создание макета для альбомной ориентации

Сверните панель LogCat. (Если вместо этого вы случайно закроете панель LogCat, ее всегда можно открыть заново командой Window ▸ Show View...)

Затем на панели Package Explorer щелкните правой кнопкой мыши на каталоге res и создайте новую папку. Присвойте ей имя layout-land.

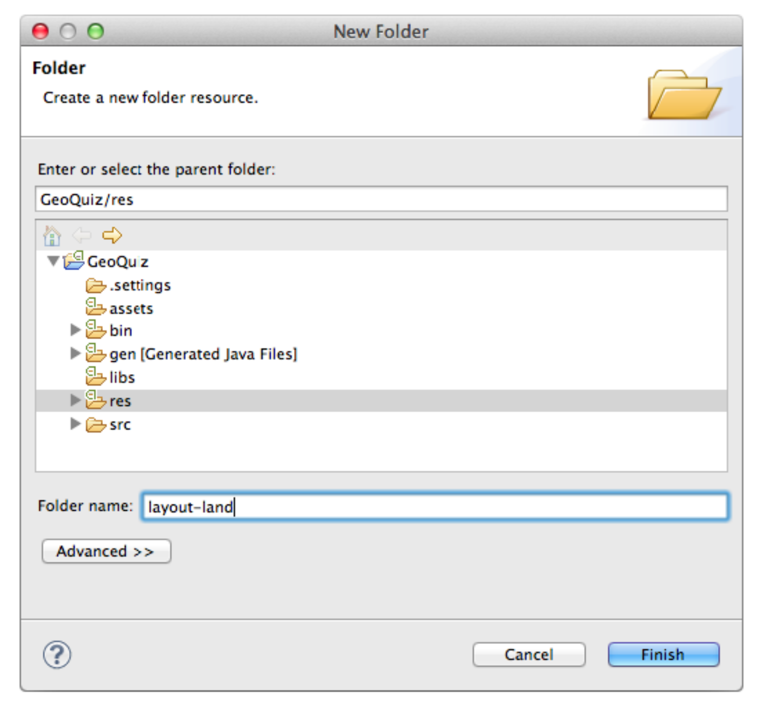


Рис. 3.11. Создание новой папки

Скопируйте файл activity_quiz.xml из res/layout/ в res/layout-land/. Теперь в приложении имеются два макета: макет для альбомной ориентации и макет по умолчанию. Оставьте имя файла без изменения. Два файла макетов должны иметь одинаковые имена, чтобы на них можно было ссылаться по одному идентификатору ресурса.

Суффикс -land — еще один пример конфигурационного квалификатора. По конфигурационным квалификаторам подкаталогов res Android определяет, какие ресурсы лучше всего подходят для текущей конфигурации устройства. Список конфигурационных квалификаторов, поддерживаемых Android, и обозначаемых ими компонентов конфигурации устройств находится по адресу *http://developer.android.com/guide/topics/resources/providing-resources.html*. Мы вернемся к использованию конфигурационных квалификаторов в главе 15.

Когда устройство находится в альбомной ориентации, Android находит и использует ресурсы в каталоге res/layout-land. В противном случае используются ресурсы по умолчанию из каталога res/layout/.

Внесем некоторые изменения в альбомный макет, чтобы он отличался от макета по умолчанию. Сводка этих изменений представлена на рис. 3.12.

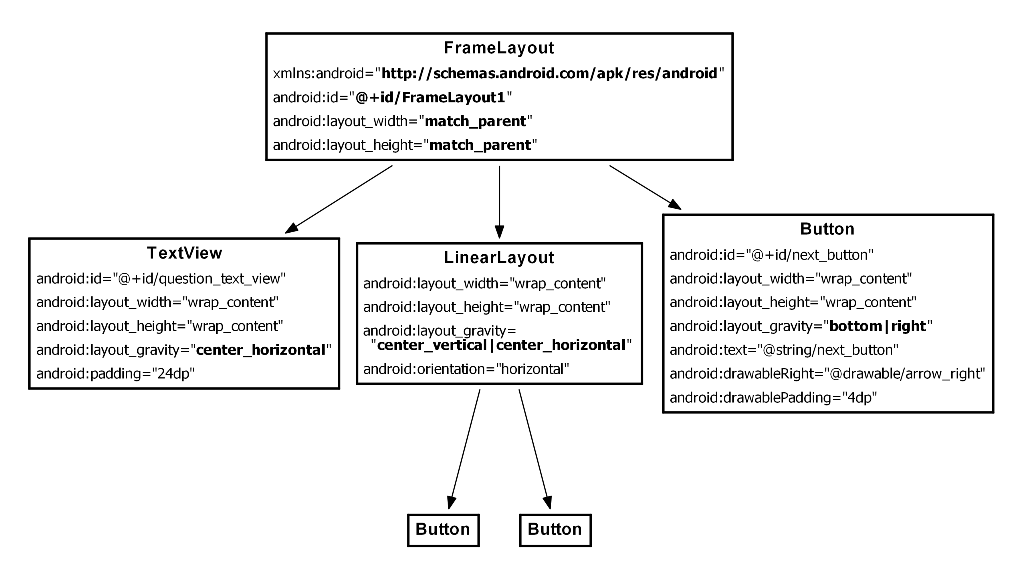


Рис. 3.12. Альтернативный макет для альбомной ориентации

Вместо `LinearLayout` будет использоваться `FrameLayout` — простейшая разновидность `ViewGroup` без определенного способа размещения потомков. В этом макете дочерние представления будут размещаться в соответствии с атрибутом `android:layout_gravity`.

Атрибут `android:layout_gravity` необходим виджетам `TextView`, `LinearLayout` и `Button`. Потомки `Button` виджета `LinearLayout` остаются без изменений.

Откройте файл layout-land/activity_quiz.xml и внесите необходимые изменения, руководствуясь рис. 3.12. Проверьте результат своей работы по листингу 3.4.

Листинг 3.4. Настройка альбомного макета (layout-land/activity_quiz.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:orientation="vertical" >

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent" >

  <TextView
    android:id="@+id/question_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:padding="24dp" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|center_horizontal"
    android:orientation="horizontal" >
    ...
  </LinearLayout>

  <Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
    />

</LinearLayout>
</FrameLayout>
```

Снова запустите GeoQuiz. Поверните устройство в альбомную ориентацию, чтобы увидеть новый макет. Конечно, в программе используется не только новый макет, но и новый экземпляр `QuizActivity`.

Поверните устройство, чтобы переключиться в книжную ориентацию. Вы увидите новый макет — и новый экземпляр `QuizActivity`.

Android выбирает наиболее подходящий ресурс за вас, но для этого он создает новую активность «с нуля». Чтобы класс `QuizActivity` вывел новый макет, необходимо снова вызвать метод `setContentView(R.layout.activity_quiz)`. А это не произойдет без повторного вызова `QuizActivity.onCreate(…)`. Соответственно

Android при повороте уничтожает текущий экземпляр `QuizActivity` и начинает все заново, чтобы обеспечить оптимальный подбор ресурсов для новой конфигурации.

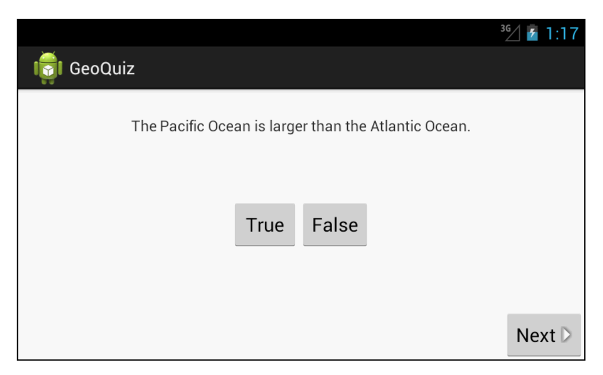


Рис. 3.13. QuizActivity в альбомной ориентации

Учтите, что Android уничтожает текущую активность и создает новую при каждом изменении конфигурации времени выполнения. Также во время выполнения могут происходит и другие изменения (например, изменение языка или доступности клавиатуры), но изменение в ориентации экрана является самым частым.

Сохранение данных между поворотами

Android очень старается предоставить альтернативные ресурсы в нужный момент. Тем не менее уничтожение и повторное создание активностей при поворотах может создать проблемы — как, например, в случае с возвратом к первому вопросу в приложении GeoQuiz.

Чтобы исправить эту ошибку, экземпляр `QuizActivity`, созданный после поворота, должен знать старое значение `mCurrentIndex`. Нам необходим механизм сохранения данных при изменении конфигурации времени выполнения (например, при поворотах). Одно из возможных решений заключается в переопределении метода `Activity`

```
protected void onSaveInstanceState(Bundle outState)
```

Обычно этот метод вызывается системой перед `onPause()`, `onStop()` и `onDestroy()`.

Реализация по умолчанию `onSaveInstanceState(…)` приказывает всем представлениям активности сохранить свое состояние в данных объекта `Bundle` — структуры, связывающей строковые ключи со значениями некоторых ограниченных типов.

Мы уже видели тип `Bundle`. Он передается методу `onCreate(Bundle)`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
```

При переопределении `onCreate(…)` вы вызываете реализацию `onCreate(…)` суперкласса активности и передаете ей только что полученный объект `Bundle`. В реализации суперкласса сохраненное состояние представлений извлекается и используется для воссоздания иерархии представлений активности.

Переопределение onSaveInstanceState(Bundle)

Метод `onSaveInstanceState(…)` можно переопределить так, чтобы он сохранял дополнительные данные в `Bundle`, а затем снова загружал их в `onCreate(…)`. Именно так мы организуем сохранение значения `mCurrentIndex` между поворотами.

Для начала добавьте в QuizActivity.java константу, которая станет ключом в сохраняемой паре «ключ-значение».

Листинг 3.5. Добавление ключа для сохраняемого значения (QuizActivity.java)

```
public class QuizActivity extends Activity {

    private static final String TAG = "QuizActivity";

    private static final String KEY_INDEX = "index";

    Button mTrueButton;
    ...
```

Теперь переопределим `onSaveInstanceState(…)` для записи значения `mCurrentIndex` в `Bundle` с использованием константы в качестве ключа.

Листинг 3.6. Переопределение onSaveInstanceState(…) (QuizActivity.java)

```
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
                updateQuestion();
            }
        });

        updateQuestion();
}

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
super.onSaveInstanceState(savedInstanceState);
Log.i(TAG, "onSaveInstanceState");
savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

Наконец, в методе `onCreate(…)` следует проверить это значение, и если оно присутствует — присвоить его `mCurrentIndex`.

Листинг 3.7. Проверка сохраненных данных в onCreate(...) (QuizActivity.java)

```
        ...

        if (savedInstanceState != null) {
      mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
    }

    updateQuestion();
}
```

Запустите GeoQuiz и нажмите кнопку Next. Сколько бы поворотов устройства вы ни выполнили, вновь созданный экземпляр QuizActivity «вспоминает» текущий вопрос.

Учтите, что для сохранения и восстановления из Bundle подходят примитивные типы и объекты, реализующие интерфейс Serializable. Создавая собственные классы, которые вы планируете сохранять в onSaveInstanceState(…), не забудьте реализовать Serializable.

Реализацию onSaveInstanceState(…) желательно протестировать — особенно если вы сохраняете и восстанавливаете объекты. Повороты тестируются легко; с тестированием ситуаций нехватки памяти дело обстоит сложнее. В конце этой главы приведена информация о том, как имитировать уничтожение вашей активности системой Android для освобождения памяти.

Снова о жизненном цикле Activity

Переопределение onSaveInstanceState(Bundle) используется не только при поворотах. Активность также может уничтожаться и в том случае, если пользователь отойдет от устройства, а Android понадобится освободить память.

Android никогда не уничтожает для освобождения памяти выполняемую активность. Чтобы активность была уничтожена, она должна находиться в приостановленном или остановленном состоянии. Если активность приостановлена или остановлена, значит, был вызван ее метод onSaveInstanceState(…).

При вызове onSaveInstanceState(…) данные сохраняются в объекте Bundle. Операционная система заносит объект Bundle в *запись активности* (activity record).

Чтобы понять, что собой представляет запись активности, добавим сохраненное состояние на схему жизненного цикла активности (рис. 3.14).

Когда ваша активность сохранена, объект Activity не существует, но объект записи активности «живет» в ОС. При необходимости операционная система может воссоздать активность по записи активности.

Следует учесть, что активность может перейти в сохраненное состояние без вызова onDestroy(). Однако вы всегда можете рассчитывать на то, что методы onPause() и onSaveInstanceState(…) были вызваны. Обычно метод onSaveInstanceState(…) переопределяется для сохранения данных в Bundle, а onPause() — для всех прочих операций.

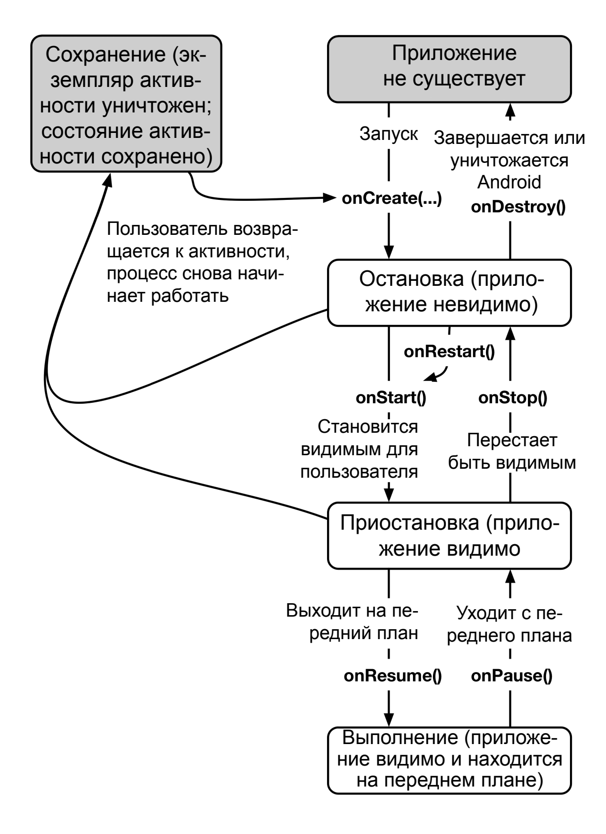


Рис. 3.14. Полный жизненный цикл активности

В некоторых ситуациях Android не только уничтожает активность, но и полностью завершает процесс приложения. Такая ситуация может возникнуть только в том случае, если пользователь не смотрит на приложение, но она возможна. Даже в этом случае запись активности продолжает существовать и позволяет быстро перезапустить активность при возвращении пользователя.

Когда же запись приложения пропадает? Когда пользователь нажимает кнопку Back, активность уничтожается — раз и навсегда. При этом запись активности теряется. Записи активности также обычно уничтожаются при перезагрузке; также возможно их уничтожение в том случае, если они слишком долго не используются.

Для любознательных: тестирование onSaveInstanceState(Bundle)

Переопределяя `onSaveInstanceState(Bundle)`, необходимо убедиться в том, что состояние сохраняется и восстанавливается так, как предполагалось. Это легко делается в эмуляторе.

Запустите виртуальное устройство. В списке приложений на устройстве найдите приложение Settings. Оно присутствует в большинстве образов системы, используемых в эмуляторе.

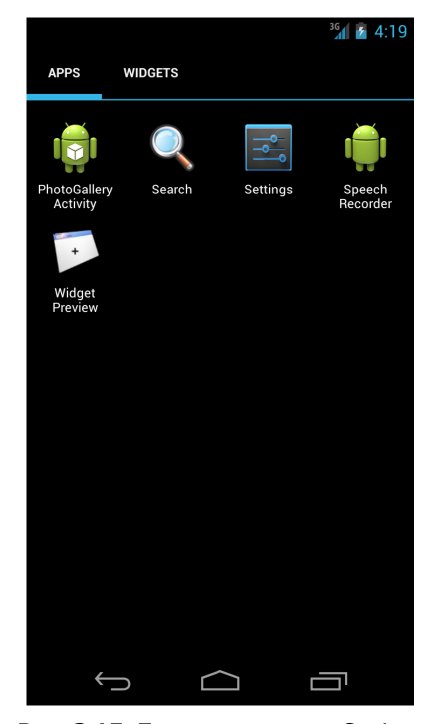


Рис. 3.15. Поиск приложения Settings

Запустите приложение Settings и выберите категорию Development options. В нее входит много разных настроек; установите флажок Don't keep activities.

Запустите приложение и нажмите кнопку Home. Это приведет к приостановке и остановке активности. Затем остановленная активность будет уничтожена так, как если бы ОС Android решила освободить занимаемую ей память. Восстановите приложение и посмотрите, было ли состояние сохранено так, как ожидалось.

Нажатие кнопки Back (вместо Home) всегда приводит к уничтожению активности независимо от того, установлен флажок в настройках разработчика или нет. Нажатие кнопки Back сообщает ОС, что пользователь завершил работу с активностью.

Чтобы выполнить тот же тест на физическом устройстве, необходимо установить на нем пакет Dev Tools. За дополнительной информацией обращайтесь по адресу *http://developer.android.com/tools/debugging/debugging-devtools.html*.

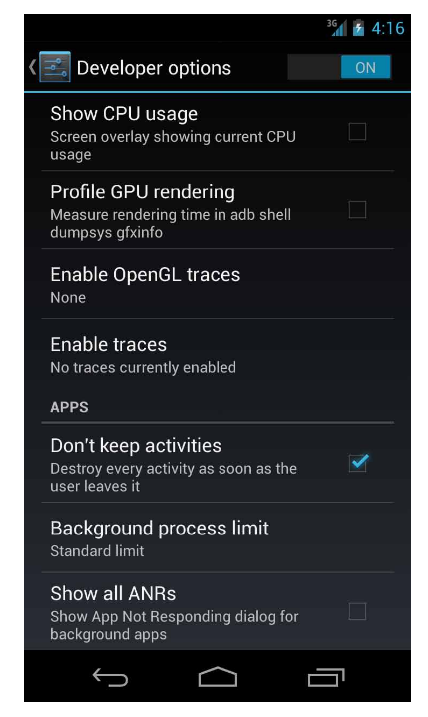


Рис. 3.16. Запрет сохранения активностей

Для любознательных: методы и уровни регистрации

Когда вы используете класс `android.util.Log` для регистрации сообщений в журнале, вы задаете не только содержимое сообщения, но и *уровень регистрации*, определяющий важность сообщения. Android поддерживает пять уровней регистрации (табл. 3.1). Каждому уровню соответствует свой метод класса `Log`. Регистрация данных в журнале сводится к вызову соответствующего метода `Log`.

Таблица 3.1. Методы и уровни регистрации

Уровень регистрации	Метод	Примечания
ERROR	Log.e(...)	Ошибки
WARNING	Log.w(...)	Предупреждения
INFO	Log.i(...)	Информационные сообщения
DEBUG	Log.d(...)	Отладочный вывод (может фильтроваться)
VERBOSE	Log.v(...)	Только для разработчиков!

Каждый метод регистрации существует в двух вариантах: один получает строковый *тег* и строку сообщения, а второй получает эти же аргументы и экземпляр `Throwable`, упрощающий регистрацию информации о конкретном исключении, которое может быть выдано вашим приложением. В листинге 3.8 представлены примеры сигнатуры методов журнала. Для сборки строк сообщений используйте стандартные средства конкатенации строк Java — или `String.format`, если их окажется недостаточно.

Листинг 3.8. Различные способы регистрации в Android

```
// Регистрация сообщения с уровнем отладки "debug"
Log.d(TAG, "Current question index: " + mCurrentIndex);

TrueFalse question;
try {
    question = mAnswerKey[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Регистрация сообщения с уровнем отладки "error"
    // вместе с трассировкой стека исключений
    Log.e(TAG, "Index was out of bounds", ex);
}
```

4 Отладка приложений Android

В этой главе вы узнаете, что делать, если в приложении скрывается ошибка. В частности, вы научитесь использовать LogCat, Android Lint и отладчик среды Eclipse.

Чтобы потренироваться в починке, нужно сначала что-нибудь сломать. В файле QuizActivity.java закомментируйте строку кода `onCreate(Bundle)`, в которой мы получаем `mQuestionTextView`.

Листинг 4.1. Из программы исключается важная строка (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
    //mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        ...
    });

    ...
}
```

Запустите GeoQuiz и посмотрите, что получится.

На рис. 4.1 показано сообщение, которое выводится при сбое приложения. В разных версиях Android используются слегка различающиеся сообщения, но все они означают одно и то же. Конечно, вы и так знаете, что случилось с приложением, но если бы не знали — было бы полезно взглянуть на приложение в другой перспективе.

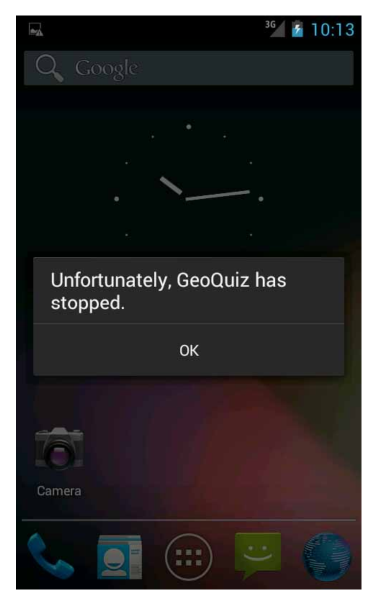


Рис. 4.1. Сбой в приложении GeoQuiz

Перспектива DDMS

Выберите в меню Eclipse команду Window ▸ Open Perspective ▸ DDMS.

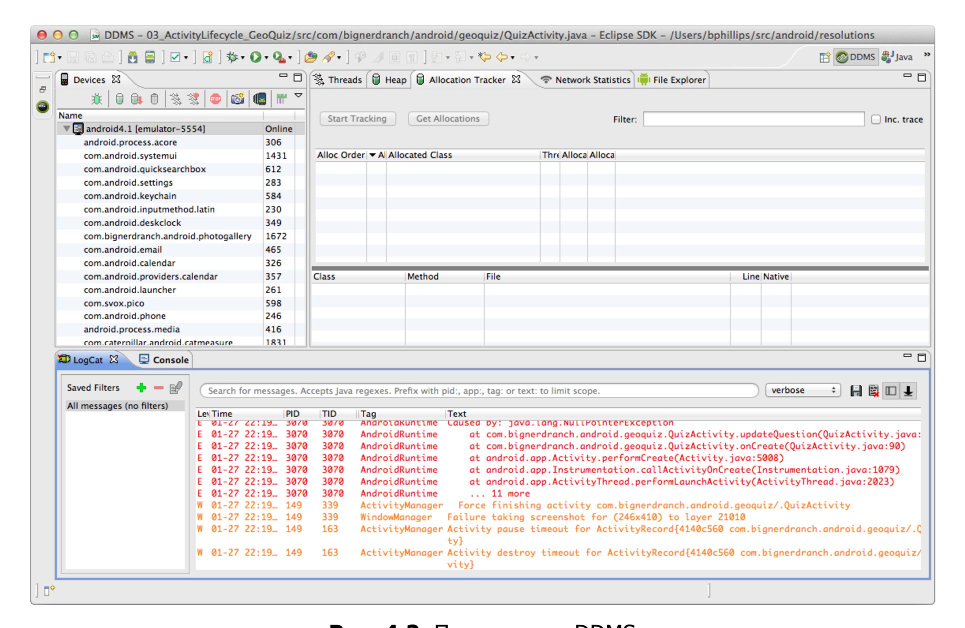


Рис. 4.2. Перспектива DDMS

Перспективой (perspective) в Eclipse называется предопределенный набор панелей. Обычно во время отладки используются одни панели, во время редактирования — другие; Eclipse объединяет такие панели в перспективу.

Перспективы определяются заранее, но жестко не фиксируются. Панели можно добавлять и удалять, а Eclipse запоминает ваш выбор. А если вам когда-нибудь захочется вернуться к исходному состоянию, достаточно выполнить команду Window ▸ Reset Perspective...

Перспектива по умолчанию, в которой вы редактировали код, называется *перспективой Java*. Перспективы, открытые в настоящее время, перечислены на кнопках в правом верхнем углу инструментальной области Eclipse. При помощи этих кнопок можно быстро переключаться между перспективами.

На рис. 4.2 изображена перспектива DDMS (Dalvik Debug Monitor Service). Механизм DDMS обеспечивает всю техническую основу отладки в Android. В перспективу DDMS входят панели LogCat и Devices.

На панели Devices представлены устройства и виртуальные устройства, подключенные к компьютеру. Обычно на этой панели решаются проблемы, относящиеся к конкретному устройству.

Например, устройство не входит в список вариантов, предлагаемых при запуске приложения? Щелкните на кнопке с треугольником в правом верхнем углу и выберите команду Reset adb. Часто после перезагрузки *adb* устройство обнаруживается системой. А может, LogCat выводит данные по другому устройству? Нет проблем — щелкните, чтобы выбрать интересующее вас устройство, а LogCat переключится на вывод данных по выбранному устройству.

Исключения и трассировка стека

Вернемся к проблеме с нашим приложением. Чтобы понять, что произошло, разверните панель LogCat. Прокрутите список и найдите текст, выделенный красным шрифтом. Это стандартный отчет об исключениях Android.

В отчете приводится исключение верхнего уровня и данные трассировки стека; затем исключение, которое привело к этому исключению, и его трассировка стека; а так далее, пока не будет найдено исключение, не имеющее причины.

Как правило, в написанном вами коде интерес представляет именно последнее исключение. В данном примере это исключение `java.lang.NullPointerException`. Строка непосредственно под именем исключения содержит начало трассировки стека. В ней указывается класс и метод, в котором произошло исключение, а также имя файла и номер строки кода. Сделайте двойной щелчок в этой строке; Eclipse открывает указанную строку кода.

В открывшейся строке программа впервые обращается к переменной `mQuestionTextView` в методе `updateQuestion()`. Имя исключения `NullPointerException` подсказывает суть проблемы: переменная не была инициализирована.

Раскомментируйте строку с инициализацией `mQuestionTextView`, чтобы исправить ошибку.

Tag	Text
dalvikvm	Not late-enabling CheckJNI (already on)
ActivityManager	Start proc com.bignerdranch.android.geoquiz for activity com.bignerdranch.android.g Activity: pid=3116 uid=10052 gids={1028}
Trace	error opening trace file: No such file or directory (2)
QuizActivity	onCreate() called
AndroidRuntime	Shutting down VM
dalvikvm	threadid=1: thread exiting with uncaught exception (group=0x40a13300)
AndroidRuntime	FATAL EXCEPTION: main
AndroidRuntime	java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch quiz/com.bignerdranch.android.geoquiz.QuizActivity}: java.lang.NullPointerException
AndroidRuntime	at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2059)
AndroidRuntime	at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2084)
AndroidRuntime	at android.app.ActivityThread.access$600(ActivityThread.java:130)
AndroidRuntime	at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1195)
AndroidRuntime	at android.os.Handler.dispatchMessage(Handler.java:99)
AndroidRuntime	at android.os.Looper.loop(Looper.java:137)
AndroidRuntime	at android.app.ActivityThread.main(ActivityThread.java:4745)
AndroidRuntime	at java.lang.reflect.Method.invokeNative(Native Method)
AndroidRuntime	at java.lang.reflect.Method.invoke(Method.java:511)
AndroidRuntime	at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:7
AndroidRuntime	at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)
AndroidRuntime	at dalvik.system.NativeStart.main(Native Method)
AndroidRuntime	Caused by: java.lang.NullPointerException
AndroidRuntime	at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.ja
AndroidRuntime	at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
AndroidRuntime	at android.app.Activity.performCreate(Activity.java:5008)
AndroidRuntime	at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
AndroidRuntime	at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2023)
AndroidRuntime	... 11 more
ActivityManager	Force finishing activity com.bignerdranch.android.geoquiz/.QuizActivity
WindowManager	Failure taking screenshot for (246x410) to layer 21010
Choreographer	Skipped 31 frames! The application may be doing too much work on its main thread.
ActivityManager	Activity pause timeout for ActivityRecord{413bbdc0 com.bignerdranch.android.geoquiz

Рис. 4.3. Исключение и трассировка стека в LogCat

Помните: когда в вашей программе возникают исключения времени выполнения, следует искать последнее исключение в LogCat и первую строку трассировки стека со ссылкой на написанный вами код. Именно здесь возникает проблема, и именно здесь следует искать ответы.

Даже если сбой происходит на неподключенном устройстве, не все потеряно. Устройство сохраняет последние строки, выводимые в журнал. Длина и срок хранения журнала зависят от устройства, но обычно можно рассчитывать на то, что результаты будут храниться минимум десять минут. Подключите устройство, вызовите перспективу DDMS в Eclipse и выберите свое устройство на панели Devices. LogCat заполняется данными из сохраненного журнала.

Диагностика ошибок поведения

Проблемы с приложениями не всегда приводят к сбоям — в некоторых случаях приложение просто начинает некорректно работать. Допустим, пользователь нажимает кнопку Next, а в приложении ничего не происходит. Такие ошибки относятся к категории ошибок поведения.

В файле QuizActivity.java внесите изменение в слушателя `mNextButton` и закомментируйте код, увеличивающий `mCurrentIndex`.

Листинг 4.2. Из программы исключается важная строка (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
            //mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
            updateQuestion();
        }
    });
    ...
}
```

Запустите GeoQuiz и нажмите кнопку Next. Ничего не происходит.

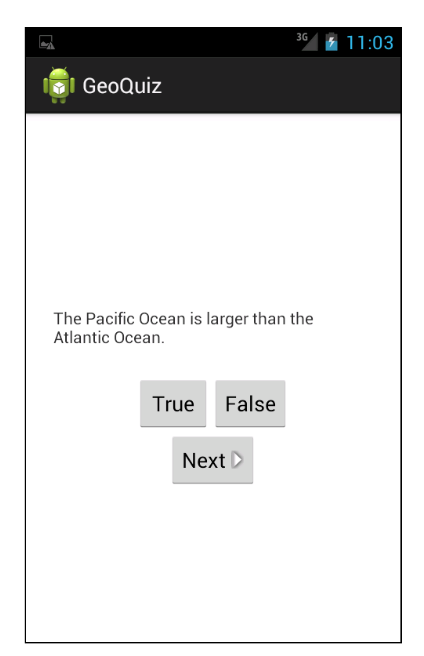


Рис. 4.4. Кнопка Next не работает

Эта ошибка коварнее предыдущей. Она не приводит к выдаче исключения, поэтому исправление ошибки не сводится к простому устранению исключения. Кроме того, некорректное поведение может быть вызвано разными причинами: то ли в программе не изменяется индекс, то ли не вызывается метод `updateQuestion()`.

Если вы не знаете причину происходящего, необходимо ее выяснить. Далее я покажу два основных приема диагностики: сохранение трассировки стека и использование отладчика для назначения точки прерывания.

Сохранение трассировки стека

Включите команду сохранения отладочного вывода в метод `updateQuestion()` класса `QuizActivity`:

Листинг 4.3. Использование Exception

```
public class QuizActivity extends Activity {
    ...

    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
          new Exception());
        int question = mAnswerKey[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
```

Версия `Log.d` с сигнатурой `Log.d(String, String, Throwable)` регистрирует в журнале все данные трассировки стека — как уже встречавшееся ранее исключение `AndroidRuntime`. По данным трассировки стека вы сможете определить, в какой момент произошел вызов `updateQuestion()`.

При вызове `Log.d(…)` вовсе не обязательно передавать перехваченное исключение. Вы можете создать новый экземпляр `Exception()` и передать его методу без инициирования исключения. В журнале будет сохранена информация о том, где было создано исключение.

Запустите приложение GeoQuiz, нажмите кнопку Next и проверьте вывод в LogCat.

Tag	Text
	eNotFoundException: /proc/net/xt_qtaguid/iface_stat_all: o irectory)
SizeAdaptiveLa…	com.android.internal.widget.SizeAdaptiveLayout@41ccc060chi cd2c30 measured out of bounds at 95px clamped to 96px
QuizActivity	Updating question text for question #0
QuizActivity	java.lang.Exception
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.updat
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.acces
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity$3.onC
QuizActivity	at android.view.View.performClick(View.java:4084)
QuizActivity	at android.view.View$PerformClick.run(View.java:16966)
QuizActivity	at android.os.Handler.handleCallback(Handler.java:615)
QuizActivity	at android.os.Handler.dispatchMessage(Handler.java:92)
QuizActivity	at android.os.Looper.loop(Looper.java:137)
QuizActivity	at android.app.ActivityThread.main(ActivityThread.java
QuizActivity	at java.lang.reflect.Method.invokeNative(Native Method
QuizActivity	at java.lang.reflect.Method.invoke(Method.java:511)
QuizActivity	at com.android.internal.os.ZygoteInit$MethodAndArgsCal
QuizActivity	at com.android.internal.os.ZygoteInit.main(ZygoteInit.
QuizActivity	at dalvik.system.NativeStart.main(Native Method)

Рис. 4.5. Результаты

Верхняя строка трассировки стека соответствует строке, в которой в журнале были зарегистрированы данные `Exception`. Следующая строка показывает, где был вызван `set`-метод — в реализации `onTextChanged(…)`. Сделайте двойной щелчок на этой строке; открывается позиция, в которой заголовку присваивается фиксированная строка. Но пока не торопитесь исправлять ошибку; сейчас мы найдем ее повторно при помощи отладчика.

Регистрация в журнале данных трассировки стека — мощный инструмент отладки, но он выводит довольно большой объем данных, которые, к тому же, содержат внутреннюю информацию о программе. Оставьте в программе несколько таких команд, и вскоре вывод LogCat превращается в невразумительную мешанину. Кроме того, по трассировке стека конкуренты могут разобраться, как работает ваш код, и похитить ваши идеи.

С другой стороны, в некоторых ситуациях нужна именно трассировка стека с информацией, показывающей, что делает ваш код. Если вы обратитесь за помощью на сайты *http://stackoverflow.com* или *forums.bignerdranch.com*, в вопрос часто желательно включить трассировку стека. Информацию можно либо скопировать прямо из LogCat, либо сохранить в текстовом файле (щелкните на кнопке с изображением дискеты в правом верхнем углу панели LogCat).

Прежде чем продолжать, закомментируйте константу `TAG` и удалите команду `log` из QuizActivity.java.

Листинг 4.4. Прощай, друг (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...

    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
          new Exception());
        int question = mAnswerKey[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
```

Закомментированная константа `TAG` подавляет предупреждение о неиспользуемой переменной. Вообще говоря, ее можно удалить, но она неожиданно может снова понадобиться для записи в журнал.

Установка точек прерывания

Попробуем найти ту же ошибку при помощи отладчика среды Eclipse. Мы установим *точку прерывания* в `updateQuestion()`, чтобы увидеть, был ли вызван этот метод. Точка прерывания останавливает выполнение программы в заданной позиции, чтобы вы могли в пошаговом режиме проверить, что происходит далее.

В файле QuizActivity.java вернитесь к методу `updateQuestion()`. В первой строке метода сделайте двойной щелчок на серой полосе слева от кода. На месте щелчка появляется синий кружок; он обозначает точку прерывания.

```
private void updateQuestion() {
    int question = mAnswerKey[mCurrentIndex].getQuestion();
    mQuestionTextView.setText(question);
}
```

Рис. 4.6. Точка прерывания

Чтобы задействовать отладчик и активизировать точку прерывания, необходимо запустить приложение в отладочном режиме (в отличие от обычного запуска). Для этого щелкните правой кнопкой мыши на проекте GeoQuiz и выберите команду `Debug As ▸ Android Application`.

Устройство сообщает, что оно ожидает подключения отладчика, а затем продолжает работу, как обычно.

Когда приложение запустится и заработает под управлением отладчика, его выполнение прерывается. Запуск GeoQuiz привел к вызову метода `QuizActivity.onCreate(Bundle)`, который вызвал `updateQuestion()`, что привело к срабатыванию точки прерывания.

Если вы впервые используете отладчик, на экране появляется большое окно со словами об открытии перспективы Debug. Щелкните на кнопке Yes, чтобы открыть перспективу Debug.

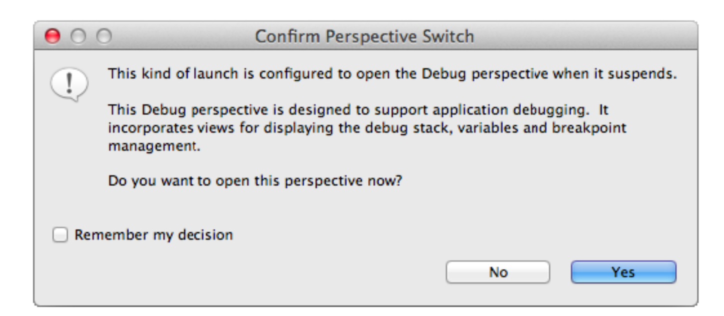


Рис. 4.7. Переключение на перспективу Debug

Eclipse открывает перспективу Debug. Центральное место в ней занимает панель редактора. В редакторе открыт файл QuizActivity.java, а в нем выделена строка с точкой прерывания, вызвавшей остановку выполнения.

Над редактором расположена панель Debug с текущей трассировкой стека.

Кнопки с желтыми стрелками в верхней части панели используются для пошагового выполнения программы. Из трассировки стека видно, что метод `updateQuestion()` был вызван из `onCreate(Bundle)`. Так как нас интересует поведение кнопки Next, нажмите кнопку Resume, чтобы продолжить выполнение программы. Затем снова нажмите кнопку Next, чтобы увидеть, активизируется ли точка прерывания (она должна активизироваться).

Теперь, когда мы добрались до интересного момента выполнения программы, можно немного осмотреться. Наверху справа найдите панель Variables. На ней можно просмотреть текущие значения объектов вашей программы. Когда эта панель открывается

в первый раз, на ней выводится только одно значение: `this` (сам объект `QuizActivity`). Щелкните на кнопке с треугольником рядом с `this` или нажмите клавишу со стрелкой вправо. Результат должен выглядеть примерно так, как на рис. 4.9.

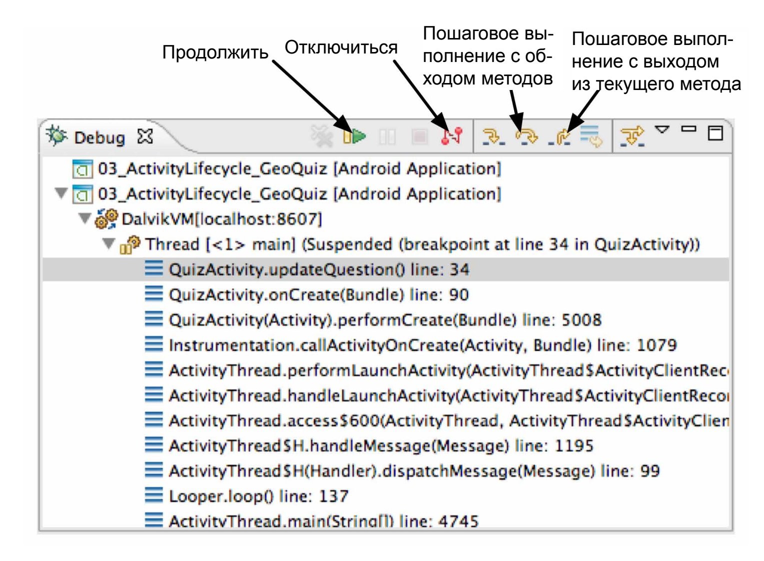


Рис. 4.8. Панель Debug

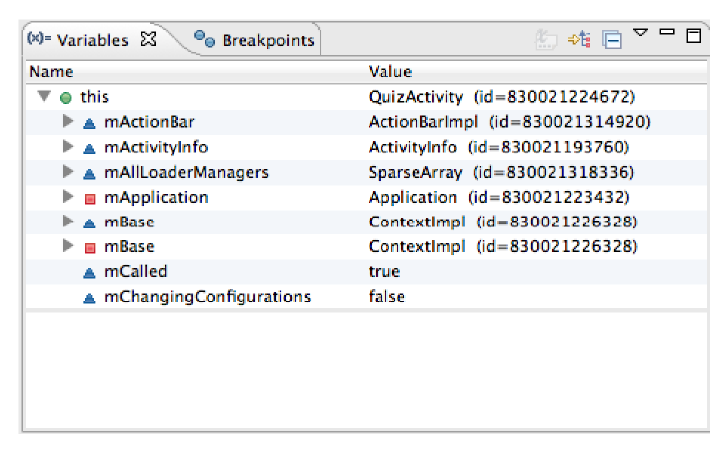


Рис. 4.9. Панель просмотра переменных

Цветные фигуры рядом с именами переменных обозначают их видимость:

- зеленый кружок — открытая (public) переменная;
- синий треугольник — переменная с видимостью по умолчанию (видимость в пределах пакета);
- желтый ромб — защищенная (protected) переменная;
- красный квадрат — закрытая (private) переменная.

В развернутом виде объект this выглядит немного устрашающе. В него включены не только переменные экземпляра, объявленные в QuizActivity, но и все переменные, объявленные в суперклассе, Activity, в суперклассе Activity, в его суперклассе и т. д.

Нас сейчас интересует только одно значение: mCurrentIndex. Прокрутите список и найдите в нем mCurrentIndex. Разумеется, переменная равна 0.

Код выглядит вполне нормально. Чтобы продолжить расследование, необходимо выйти из метода. Щелкните на кнопке, расположенной справа от кнопки Step Over (если навести на нее указатель мыши, выводится подсказка Step Return). (Из-за вызова вспомогательного метода access$1(QuizActivity) на кнопке Step Return придется щелкнуть дважды.)

Взгляните на панель редактора — управление передано слушателю OnClickListener кнопки mNextButton, в точку непосредственно после вызова updateQuestion(). Удобно, что и говорить.

Ошибку нужно исправить, но прежде чем вносить какие-либо изменения в код, необходимо прервать отладку приложения. Это можно сделать двумя способами: либо остановив программу, либо простым отключением отладчика. Чтобы остановить программу, следует выбрать процесс вашей программы на панели DDMS Devices и щелкнуть на кнопке с красным стоп-сигналом. Вариант с отключением обычно проще: щелкните на кнопке Disconnect (см. рис. 4.8). Затем снова переключитесь на перспективу Java и верните OnClickListener в прежнее состояние.

Листинг 4.5. Возвращение к исходному состоянию (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
            mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
            updateQuestion();
        }
    });
    ...
}
```

Также необходимо убрать точку прерывания. Найдите панель Breakpoints рядом с панелью Variables. (Если эта панель не отображается, откройте ее командой Window ▸ Show View ▸ Breakpoints.) Выделите точку прерывания и щелкните на темно-серой кнопке X в верхней части панели.

Мы рассмотрели два способа поиска проблемной строки кода: сохранение в журнале трассировки стека и установка точки прерывания в отладчике. Какой способ лучше? Каждый находит свои применения, и скорее всего, один из них станет для вас основным.

Преимущество трассировки стека заключается в том, что трассировки из нескольких источников просматриваются в одном журнале. С другой стороны, чтобы получить новую информацию о программе, вы должны добавить новые команды регистрации, заново построить приложение, развернуть его и добраться до нужной точки. С отладчиком работать проще. Запустив приложение с подключенным отладчиком (команда Debug As ▸ Android Application), вы сможете установить точку прерывания во время работы приложения, а потом поэкспериментировать для получения информации сразу о разных проблемах.

Прерывания по исключениям

Если вам недостаточно этих решений, вы также можете использовать отладчик для перехвата исключений. Вернитесь к коду приложения и снова закомментируйте строку с присваиванием `mQuestionTextView`. Выполните команду Run ▸ Add Java Exception Breakpoint..., чтобы вызвать диалоговое окно прерывания по исключению.

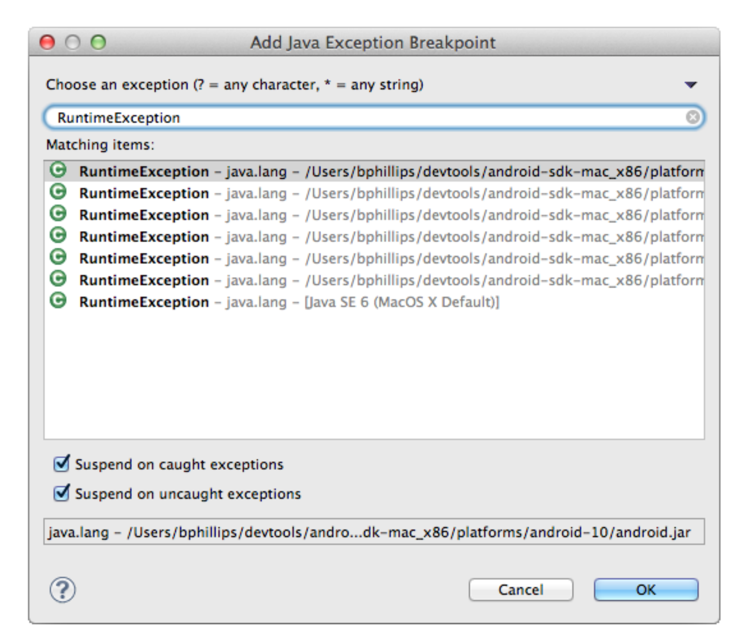


Рис. 4.10. Установка точки прерывания по исключению

Диалоговое окно позволяет установить точку прерывания, которая срабатывает при инициировании исключения, где бы оно ни произошло. Прерывания можно ограничить только неперехваченными исключениями или же применить их как к перехваченным, так и неперехваченным исключениям.

В Android большинство исключений перехватывается инфраструктурой, которая выводит приведенное ранее диалоговое окно и завершает процесс. Это означает, что обычно для прерываний по исключениям следует выбирать режим перехваченных исключений Suspend on caught exceptions.

Теперь выберите, какие исключения должны перехватываться. Введите имя RuntimeException и выберите любые из предлагаемых вариантов. RuntimeException является суперклассом NullPointerException, ClassCastException и других проблем времени выполнения, с которыми вы можете столкнуться, поэтому этот вариант удобен своей универсальностью.

Впрочем, чтобы он перехватывал типы субклассов исключений, необходимо сделать еще кое-что. Переключитесь на перспективу Debug и взгляните на панель Breakpoints. Вы увидите в ней новую точку прерывания для RuntimeException. Щелкните на ней и выберите команду Subclasses of this exception, чтобы отладчик прерывал работу и при получении NullPointerException.

Включите отладку GeoQuiz. На этот раз отладчик сразу переходит к строке, в которой произошло исключение, — замечательно.

Учтите, что если эта точка прерывания останется включенной во время отладки, она может сработать на инфраструктурном коде или в других местах, на которые вы не рассчитывали. Не забудьте отключить ее, если она не используется.

File Explorer

Перспектива DDMS предоставляет другие эффективные средства контроля за работой приложения, включая очень удобный файловый менеджер. Выберите панель File Explorer в группе вкладок справа.

Threads | Heap | Allocation Tracker | File Explorer

Name	Size	Date	Time	Permissions	Info
data		2011-12-21	11:28	drwxrwx--x	
mnt		2012-05-07	09:54	drwxrwxr-x	
system		2012-04-05	11:21	drwxr-xr-x	
app		2012-04-05	11:20	drwxr-xr-x	
bin		2012-04-05	11:20	drwxr-xr-x	
build.prop	2058	2012-04-05	11:21	-rw-r--r--	
etc		2012-04-05	11:21	drwxr-xr-x	
fonts		2012-04-05	11:20	drwxr-xr-x	
framework		2012-04-05	11:20	drwxr-xr-x	
lib		2012-04-05	11:21	drwxr-xr-x	
lost+found		1969-12-31	19:00	drwxr-xr-x	
media		2011-12-20	11:32	drwxr-xr-x	
modules		2012-04-05	11:21	drwxr-xr-x	
recovery-from-boot.p	323002	2008-08-01	08:00	-rw-r--r--	

Рис. 4.11. Файловый менеджер

В файловом менеджере можно просматривать файловую систему устройства, загружать и отправлять файлы. На физическом устройстве вы не сможете заглянуть в каталог /data, но в эмуляторе это возможно — следовательно, вы можете просматривать область хранения данных своего приложения. В новейшей версии Android эта область находится в каталоге /data/data/*[имя пакета]*.

Name	Size	Date	Time	Permissions	Info
com.android.sharedstoragebackup		2012-07-25	15:26	drwxr-x--x	
com.android.smoketest		2012-07-24	19:56	drwxr-x--x	
com.android.smoketest.tests		2012-07-24	19:56	drwxr-x--x	
com.android.soundrecorder		2012-07-24	19:56	drwxr-x--x	
com.android.speechrecorder		2012-07-24	19:56	drwxr-x--x	
com.android.systemui		2012-07-24	19:56	drwxr-x--x	
com.android.vpndialogs		2012-07-24	19:56	drwxr-x--x	
com.android.wallpaper.livepicker		2012-07-24	19:56	drwxr-x--x	
com.android.widgetpreview		2012-07-24	19:56	drwxr-x--x	
com.bignerdranch.android.criminalintent		2012-11-14	15:02	drwxr-x--x	
com.bignerdranch.android.draganddraw		2012-08-01	15:33	drwxr-x--x	
com.bignerdranch.android.geoquiz		2013-01-14	00:37	drwxr-x--x	
com.bignerdranch.android.hellomoon		2012-08-06	14:09	drwxr-x--x	

Рис. 4.12. Каталог данных GeoQuiz (в эмуляторе)

Пока здесь смотреть не на что, но в главе 17 файловый менеджер будет использоваться для просмотра файлов, записанных в закрытое хранилище.

Особенности отладки Android

Обычно процесс отладки в Android не отличается от обычной отладки кода Java. Тем не менее у вас могут возникнуть проблемы в областях, специфических для Android (например, ресурсы), о которых компилятор Java ничего не знает.

Android Lint

На помощь приходит Android Lint — *статический анализатор* кода Android. Статические анализаторы проверяют код на наличие дефектов, не выполняя его. Android Lint использует свое знание инфраструктуры Android для проверки кода и выявления проблем, которые компилятор обнаружить не может. Как правило, к рекомендациям Android Lint стоит прислушиваться.

В главе 6 вы увидите, как Android Lint выдает предупреждение о проблеме совместимости. Кроме того, Android Line может выполнять проверку типов для объектов, определенных в XML. Попробуйте включить в `QuizActivity` следующую ошибку преобразования типов.

Листинг 4.6. Ошибка в указании типа (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
```

продолжение ⇗

Листинг 4.6 (*продолжение*)

```
    mTrueButton = (Button)findViewById(R.id.question_text_view);

    ...
}
```

Из-за указания неверного идентификатора ресурса код попытается преобразовать `TextView` в `Button` во время выполнения, что приведен к исключению неправильного преобразования типа. Компилятор Java не видит проблем в этом коде, но Android Lint обнаружит ошибку еще до запуска приложения.

На панели Package Explorer щелкните правой кнопкой мыши на проекте GeoQuiz и выберите команду Android Tools ▸ Run Lint: Check for Common Errors; появляется панель Lint Warnings (рис. 4.13).

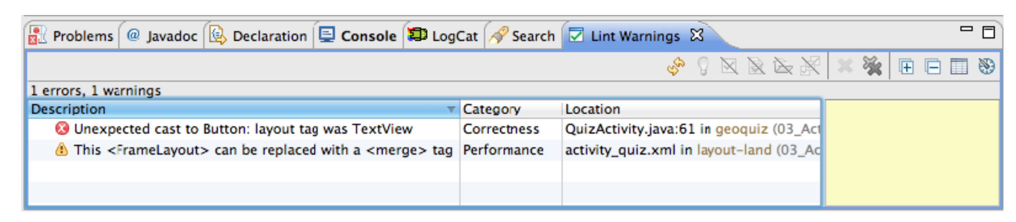


Рис. 4.13. Предупреждения Lint

В данном случае Android Lint выдает одну ошибку и одно предупреждение. Ошибка нам уже известна: с неверным идентификатором ресурса преобразование к типу `Button` завершится неудачей. Исправьте ошибку в `onCreate(Bundle)`.

Листинг 4.7. Исправление простой ошибки (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.question_text_view);
    mTrueButton = (Button)findViewById(R.id.true_button);

    ...
}
```

Предупреждение от Android Lint не столь вразумительно. В нем рекомендуется использовать тег `merge` для layout-land/activity_quiz.xml. К сожалению, Android Lint здесь ошибается. Не заменяйте `FrameLayout` тегом `merge`, потому что `FrameLayout` используется для конкретного позиционирования ваших виджетов.

Проблемы с классом R

Всем известны ошибки построения, которые возникают из-за ссылок на ресурсы до их добавления (или удаления ресурсов, на которые ссылаются другие файлы). Обычно повторное сохранение файла после добавления ресурса или удаления ссылки приводит к тому, что Eclipse проводит построение заново, а проблемы исчезают.

Однако иногда такие ошибки остаются или появляются ниоткуда. Если вы столкнетесь с подобной ситуацией, попробуйте принять следующие меры.

Запустите Android Lint

Выполните команду Window ▸ Run Android Lint. Часто запуск Line налаживает нарушенные связи между составляющими проекта.

Выполните чистку проекта

Выполните команду Project ▸ Clean. Eclipse строит проект заново, а результат построения нередко избавляется от ошибок.

Проверьте разметку XML в файлах ресурсов

Если файл R.java не был сгенерирован при последнем построении, то все ссылки на ресурс будут сопровождаться ошибками. Часто ошибки вызваны опечатками в разметке одного из файлов XML. Разметка макета не проверяется, поэтому среда не сможет привлечь ваше внимание к опечаткам в таких файлах. Если вы найдете ошибку и заново сохраните файл, код R.java будет сгенерирован заново.

Удаление каталога gen

Если вам не удается заставить Eclipse сгенерировать новый файл R.java, попробуйте удалить весь каталог gen. При повторном построении Eclipse создает новый каталог gen с работоспособным классом R.

Если у вас все еще остаются проблемы с ресурсами (или иные проблемы), отдохните и просмотрите сообщения об ошибках и файлы макета «на свежую голову». В запале легко пропустить ошибку. Также проверьте все ошибки и предупреждения Android Lint. При спокойном повторном рассмотрении сообщений нередко выявляются ошибки или опечатки.

Наконец, если вы зашли в тупик или у вас возникли другие проблемы с Eclipse, обратитесь к архивам *http://stackoverflow.com* или посетите форум книги по адресу *http://forums.bignerdranch.com*.

5 Вторая активность

В этой главе мы добавим в приложение GeoQuiz вторую активность. Как было сказано ранее, активность управляет информацией на экране; новая активность добавит в приложение второй экран, на котором пользователю будет предложено увидеть ответ на текущий вопрос.

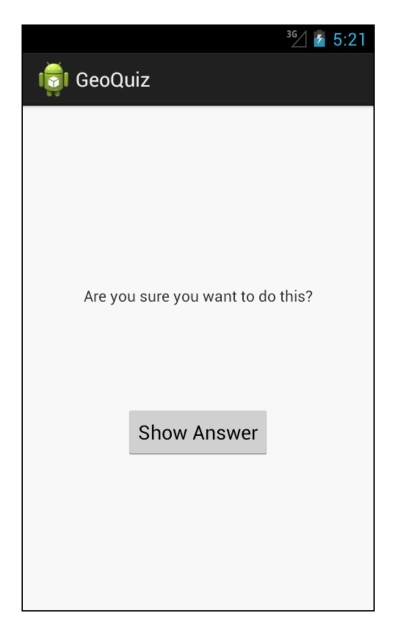


Рис. 5.1. CheatActivity позволяет подсмотреть ответ на вопрос

Если пользователь решает просмотреть ответ, а затем возвращается к `QuizActivity` и отвечает на вопрос, он получает новое сообщение.

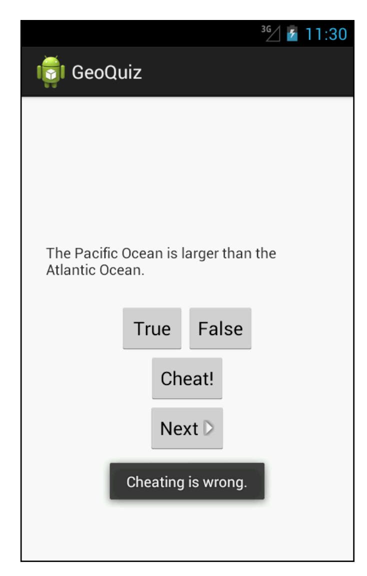


Рис. 5.2. QuizActivity знает, что вы жульничаете

Почему эта задача является хорошим упражнением по программированию Android? Потому что вы научитесь:

- создавать новую активность и новый макет без помощи мастера новых приложений;
- запускать активность из другой активности (вы приказываете ОС создать экземпляр активности и вызвать его метод `onCreate(Bundle)`);
- передавать данные между родительской (запускающей) и дочерней (запущенной) активностью.

Создание второй активности

В этой главе нам предстоит много сделать. Мы начнем с создания нового макета для `CheatActivity`, после чего создадим сам класс `CheatActivity`.

Но сначала откройте файл strings.xml и добавьте строки, необходимые для этой главы.

Листинг 5.1. Добавление строк (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  ...
  <string name="question_asia">Lake Baikal is the world\'s oldest and deepest
      freshwater lake.</string>
  <string name="cheat_button">Cheat!</string>
  <string name="warning_text">Are you sure you want to do this?</string>
  <string name="show_answer_button">Show Answer</string>
  <string name="judgment_toast">Cheating is wrong.</string>

</resources>
```

Создание нового макета

На рис. 5.1 показано, как должно выглядеть представление `CheatActivity`. Определения виджетов приведены на рис. 5.3.

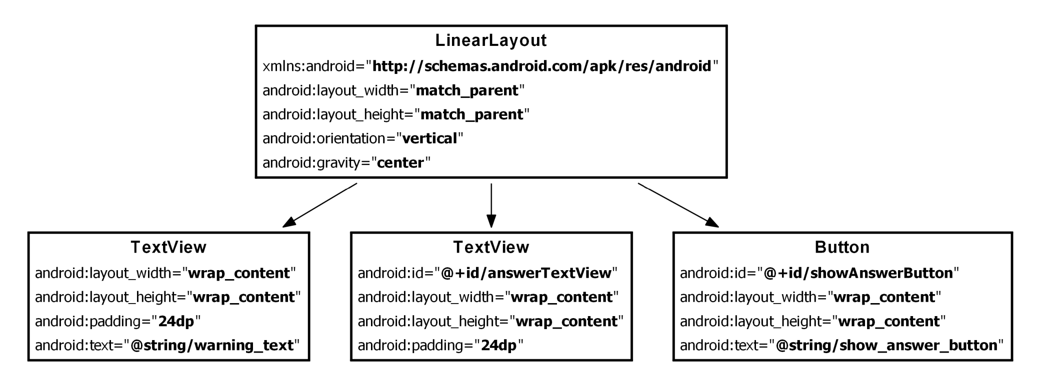


Рис. 5.3. Схема макета CheatActivity

Чтобы создать файл макета, щелкните правой кнопкой мыши на каталоге res/layout на панели Package Explorer и выберите команду New ▸ Other... В папке Android найдите и выберите строку Android XML Layout File (рис. 5.4). Щелкните на кнопке Next.

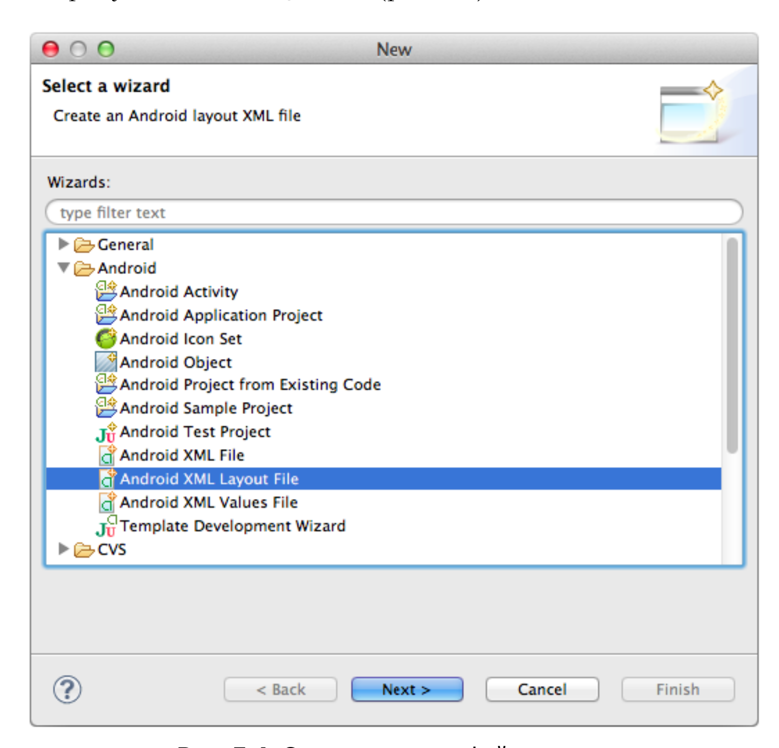


Рис. 5.4. Создание нового файла макета

В открывшемся диалоговом окне введите имя файла макета activity_cheat.xml и выберите корневой элемент `LinearLayout`. Щелкните на кнопке Finish.

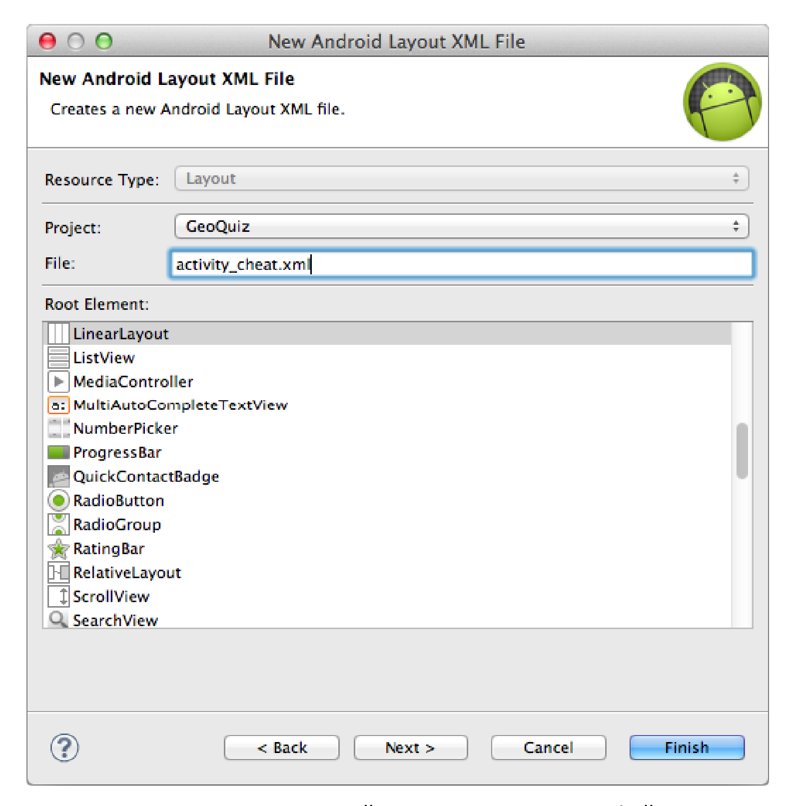


Рис. 5.5. Ввод имени и настройка параметров нового файла макета

Когда файл откроется, перейдите к разметке XML. Обратите внимание: мастер включил в начало файла строку

```
<?xml version="1.0" encoding="utf-8"?>
```

Эта строка не является обязательной для файлов макетов XML, но она все равно добавляется в файлы, создаваемые некоторыми источниками (включая этого мастера создания макета).

(Кстати говоря, если вы терпеть не можете графические интерфейсы, использовать мастера не обязательно. Просто создайте новый файл с именем fragment_crime.xml в каталоге res/layout и обновите папку res/layout, чтобы среда Eclipse нашла этот файл. Это относится к большинству мастеров Eclipse; ничто не мешает вам создавать файлы XML и файлы классов Java тем способом, который вы предпочитаете. Единственный мастер, без которого вам не обойтись, — это мастер новых приложений Android).

Мастер создания макета сгенерировал корневой элемент `LinearLayout` за вас. Остается лишь добавить атрибут `android:gravity` и трех потомков.

Попробуйте создать разметку XML для activity_cheat.xml по образцу рис. 5.3. После главы 8 в тексте вместо длинных фрагментов XML будут приводиться только схемы макетов вроде рис. 5.3, так что вам стоит освоить самостоятельное создание XML макетов. Сравните свою работу с листингом 5.2.

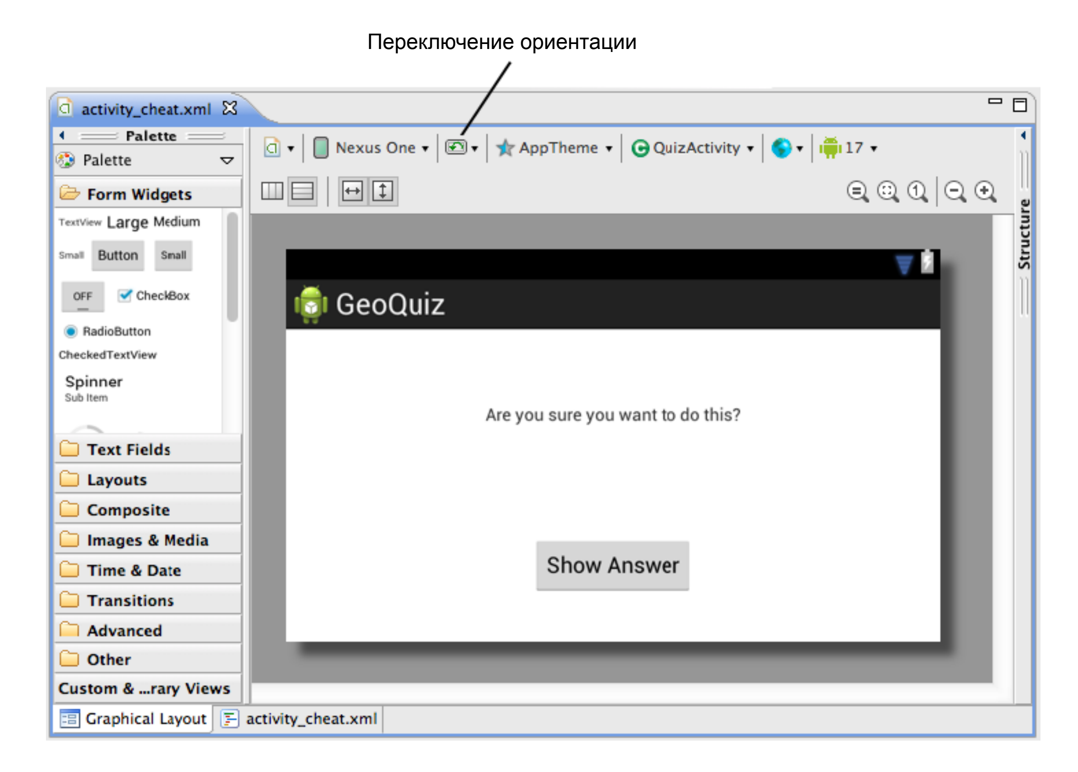


Рис. 5.6. Предварительный просмотр activity_cheat.xml в альбомной ориентации

Листинг 5.2. Заполнение макета второй активности (activity_cheat.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:gravity="center">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/warning_text_view" />

  <TextView
    android:id="@+id/answerTextView"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp" />

    <Button
        android:id="@+id/showAnswerButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/show_answer_button" />

</LinearLayout>
```

Сохраните файл, переключитесь в графический конструктор макетов и посмотрите, как выглядит результат.

Мы не будем создавать для activity_cheat.xml альтернативный макет с альбомной ориентацией, однако существует возможность увидеть, как макет по умолчанию будет отображаться в альбомном режиме.

В графическом конструкторе найдите на панели инструментов над панелью предварительного просмотра кнопку, на которой изображено устройство с зеленой стрелкой. Щелкните на этой кнопке, чтобы изменить ориентацию макета.

Макет по умолчанию достаточно хорошо смотрится в обеих ориентациях, можно переходить к созданию субкласса активности.

Создание нового субкласса активности

На панели Package Explorer щелкните правой кнопкой мыши на пакете com.bignerdranch.android.geoquiz и выберите New ▸ Class.

В открывшемся диалоговом окне введите имя класса `CheatActivity`. В поле Superclass: введите имя суперкласса `android.app.Activity` (рис. 5.7).

Щелкните на кнопке Finish; Eclipse открывает файл CheatActivity.java в редакторе.

Добавьте реализацию `onCreate(…)`, которая передает идентификатор ресурса макета, определяемого в activity_cheat.xml, при вызове `setContentView(…)`.

Листинг 5.3. Переопределение onCreate(…) (CheatActivity.java)

```
public class CheatActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);
    }

}
```

Со временем функциональность метода `onCreate(…)` в `CheatActivity` будет расширена. А пока давайте перейдем к следующему шагу: объявлению `CheatActivity` в манифесте приложения.

Рис. 5.7. Создание суперкласса CheatActivity

Объявление активностей в манифесте

Манифест (manifest) представляет собой файл XML с метаданными, описывающими ваше приложение для ОС Android. Файл манифеста всегда называется AndroidManifest.xml и располагается в корневом каталоге вашего проекта.

На панели Package Explorer найдите и откройте AndroidManifest.xml в корневом каталоге проекта. Не обращая внимания на графический редактор, выберите вкладку AndroidManifest.xml в нижней части панели редактора.

Каждая активность приложения должна быть объявлена в манифесте, чтобы она стала доступной для ОС. Когда вы использовали мастер новых приложений для создания `QuizActivity`, мастер объявил активность за вас. Для `CheatActivity` вам придется сделать это самостоятельно.

Включите в файл AndroidManifest.xml объявление `CheatActivity` (листинг 5.4).

Листинг 5.4. Объявление CheatActivity в манифесте (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.geoquiz"
```

```
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.bignerdranch.android.geoquiz.QuizActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".CheatActivity"
      android:label="@string/app_name" />
  </application>

</manifest>
```

Атрибут `android:name` является обязательным. Точка в начале значения атрибута сообщает ОС, что класс этой активности находится в пакете, который задается атрибутом `package` в элементе `manifest` в начале файла.

Манифест содержит много интересной информации, но сейчас мы хотим как можно быстрее организовать работу `CheatActivity`. Другие части манифеста будут рассматриваться позднее.

Добавление кнопки Cheat в QuizActivity

Итак, пользователь должен нажать кнопку в `QuizActivity`, чтобы вызвать на экран экземпляр `CheatActivity`. Следовательно, мы должны включить новые кнопки в layout/activity_quiz.xml и layout-land/activity_quiz.xml.

В макете по умолчанию добавьте новую кнопку как прямого потомка корневого элемента `LinearLayout`. Ее определение должно непосредственно предшествовать кнопке Next.

Листинг 5.5. Добавление кнопки Cheat! в макет по умолчанию (layout/activity_quiz.xml)

```
    ...
  </LinearLayout>

  <Button
    android:id="@+id/cheat_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

продолжение ⇗

Листинг 5.5 (*продолжение*)

```
    android:text="@string/cheat_button" />

  <Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button" />

</LinearLayout>
```

В альбомном макете новая кнопка размещается внизу и по центру корневого элемента `FrameLayout`.

Листинг 5.6. Добавление кнопки Cheat! в альбомный макет (layout-land/activity_quiz.xml)

```
    ...
  </LinearLayout>

  <Button
    android:id="@+id/cheat_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|center"
    android:text="@string/cheat_button" />

  <Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp" />

</FrameLayout>
```

Сохраните файлы макетов и откройте QuizActivity.java. Добавьте переменную, получите ссылку и назначьте заглушку `View.onClickListener` для кнопки Cheat!.

Листинг 5.7. Подключение кнопки Cheat! (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private Button mNextButton;
    private Button mCheatButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                // Запуск CheatActivity
            }
        });

        updateQuestion();
    }

    ...
}
```

Теперь можно переходить к запуску `CheatActivity`.

Запуск активности

Чтобы запустить одну активность из другой, проще всего воспользоваться методом `Activity`:

```
public void startActivity(Intent intent)
```

Напрашивается предположение, что `startActivity(…)` является методом класса, который должен вызываться для запускаемого субкласса `Activity`. Тем не менее, это не так. Когда активность вызывает `startActivity(…)`, этот вызов передается ОС.

А точнее, он передается компоненту ОС, который называется `ActivityManager`. `ActivityManager` создает экземпляр `Activity` и вызывает его метод `onCreate(…)`.

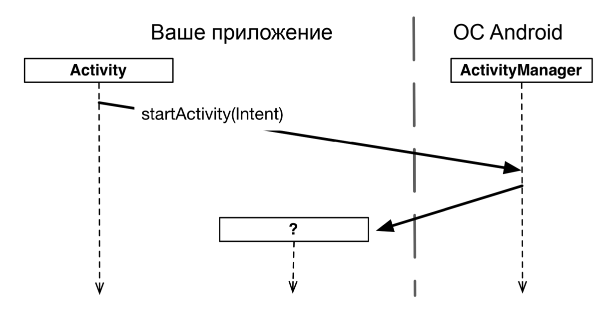


Рис. 5.8. Запуск активности

Откуда `ActivityManager` узнает, какую активность следует запустить? Эта информация передается в параметре `Intent`.

Передача информации через интенты

Интент (intent) — объект, который может использоваться компонентом для взаимодействия с ОС. Пока что из компонентов нам встречались только активности, но еще существуют службы (services), широковещательные приемники (broadcast receivers) и поставщики контента (content providers).

Интенты представляют собой многоцелевые средства передачи информации, а класс Intent предоставляет разные конструкторы в зависимости от того, для чего должен использоваться интент.

В данном случае интент сообщает ActivityManager, какую активность следует запустить, поэтому мы используем следующий конструктор:

```
public Intent(Context packageContext, Class<?> cls)
```

Объект Class задает активность, которая должна быть запущена ActivityManager. Объект Context сообщает ActivityManager, в каком пакете находится объект Class.

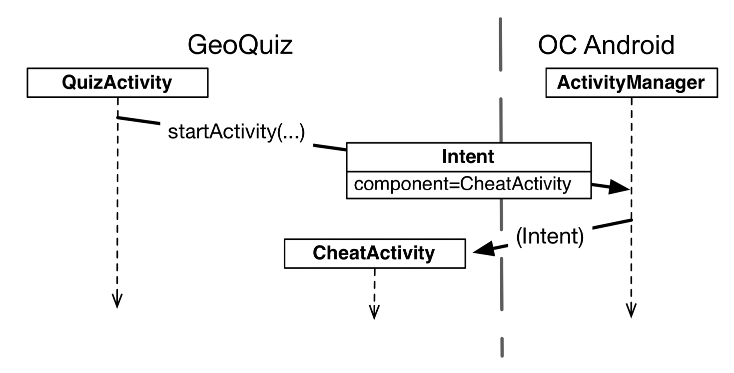


Рис. 5.9. Интент: передача ActivityManager информации о том, что нужно сделать

В слушателе mCheatButton создайте объект Intent, включающий класс CheatActivity, а затем передайте его при вызове startActivity(Intent) (листинг 5.8).

Листинг 5.8. Запуск CheatActivity (QuizActivity.java)

```
        ...

        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new Intent(QuizActivity.this, CheatActivity.class);
                startActivity(i);
            }
        });

        updateQuestion();
    }
```

Прежде чем запускать активность, ActivityManager ищет в манифесте пакета объявление с именем, соответствующим заданному объекту Class. Если такое объявление будет найдено, активность запускается — все хорошо. Если объявление не найдено, выдается исключение ActivityNotFoundException. Вот почему все активности должны объявляться в манифесте.

Интенты явные и неявные

При создании объекта Intent с объектом Class и Context вы создаете *явный* (explicit) интент. Явные интенты используются для запуска активностей в приложениях.

Может показаться странным, что две активности внутри приложения должны взаимодействовать через компонент ActivityManager, находящийся вне приложения. Тем не менее такая схема позволяет активности одного приложения легко работать с активностью другого приложения.

Когда активность в вашем приложении должна запустить активность в другом приложении, вы создаете *неявный* (implicit) интент. Использование неявных интентов рассматривается в главе 21.

Запустите GeoQuiz. Нажмите кнопку Cheat!; на экране появляется экземпляр новой активности.

Теперь нажмите кнопку Back. Активность CheatActivity уничтожается, а вы возвращаетесь к QuizActivity.

Передача данных между активностями

Итак, в нашем приложении действуют активности QuizActivity и CheatActivity, и мы можем подумать о передаче данных между ними. На рис. 5.10 показано, какие данные будут передаваться между двумя активностями.

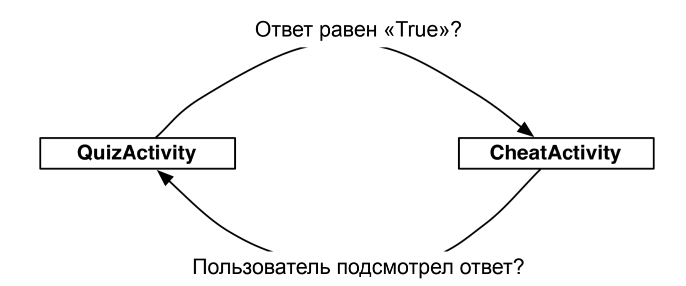


Рис. 5.10. Обмен данными между QuizActivity и CheatActivity

QuizActivity передает CheatActivity ответ на текущий вопрос при запуске CheatActivity.

Когда пользователь нажимает кнопку Back, чтобы вернуться к QuizActivity, экземпляр CheatActivity уничтожается. В последний момент он передает QuizActivity информацию о том, подсмотрел ли пользователь правильный ответ.

Начнем с передачи данных от QuizActivity к CheatActivity.

Дополнения интентов

Чтобы сообщить `CheatActivity` ответ на текущий вопрос, мы будем передавать значение

```
mAnswerKey[mCurrentIndex].isTrueQuestion();
```

Значение будет передаваться в виде *дополнения* (extra) объекта `Intent`, передаваемого `startActivity(Intent)`.

Дополнения представляют собой произвольные данные, которые вызывающая активность может передать вместе с интентом. ОС направляет интент активности получателю, которая обращается к дополнению и извлекает данные.

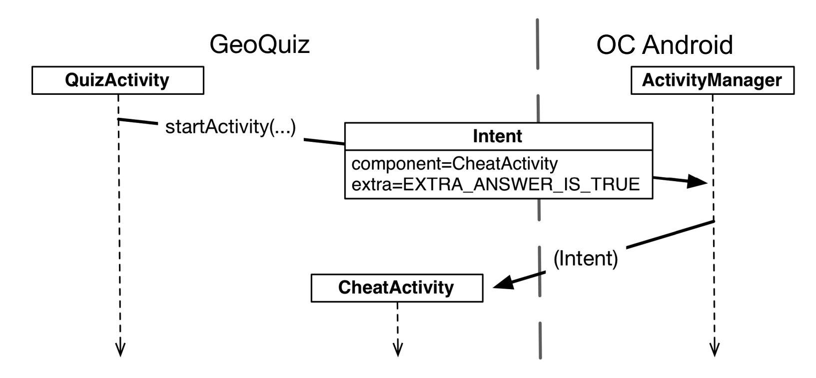


Рис. 5.11. Дополнения интентов: взаимодействие с другими активностями

Дополнение представляет собой пару «ключ-значение» наподобие той, которая использовалась для сохранения значения `mCurrentIndex` в `QuizActivity.onSaveInstanceState(Bundle)`.

Для включения дополнений в интент используется метод `Intent.putExtra(…)` — а точнее, метод

```
public Intent putExtra(String name, boolean value)
```

Метод `Intent.putExtra(…)` существует в нескольких разновидностях, но он всегда получает два аргумента. В первом аргументе всегда передается ключ `String`, а во втором — значение того или иного типа.

Добавьте в CheatActivity.java ключ для дополнения.

Листинг 5.9. Добавление константы (CheatActivity.java)

```
public class CheatActivity extends Activity {

    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";

    ...
```

Активность может запускаться из нескольких разных мест, поэтому ключи для дополнений должны определяться в активностях, которые читают и используют их. Как видно из листинга 5.9, уточнение дополнения именем пакета предотвращает конфликты имен с дополнениями других приложений.

Вернемся к `QuizActivity` и включим дополнение в интент.

Листинг 5.10. Включение дополнения в интент (QuizActivity.java)

```
        ...
        mCheatButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new Intent(QuizActivity.this, CheatActivity.class);
                boolean answerIsTrue = mAnswerKey[mCurrentIndex].isTrueQuestion();
                i.putExtra(CheatActivity.EXTRA_ANSWER_IS_TRUE, answerIsTrue);
                startActivity(i);
            }
        });

        updateQuestion();
    }
```

В нашей ситуации достаточно одного дополнения, но при необходимости можно добавить в `Intent` несколько дополнений.

Для чтения значения из дополнения используется метод

```
public boolean getBooleanExtra(String name, boolean defaultValue)
```

Первый аргумент `getBooleanExtra(…)` содержит имя дополнения, а второй — ответ по умолчанию, если ключ не найден.

В `CheatActivity` прочитайте значение из дополнения и сохраните его в переменной.

Листинг 5.11. Использование дополнения (CheatActivity.java)

```
public class CheatActivity extends Activity {
    ...

    private boolean mAnswerIsTrue;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
    }
}
```

Обратите внимание: `Activity.getIntent()` возвращает объект `Intent`, который был передан в `startActivity(Intent)`.

Наконец, добавьте в `CheatActivity` код, обеспечивающий использование прочитанного значения в виджете `TextView` ответа и кнопке Show Answer.

Листинг 5.12. Добавление функциональности подсматривания ответов (CheatActivity.java)

```
public class CheatActivity extends Activity {
    ...
    private boolean mAnswerIsTrue;

    private TextView mAnswerTextView;
    private Button mShowAnswer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);

        mAnswerTextView = (TextView)findViewById(R.id.answerTextView);

        mShowAnswer = (Button)findViewById(R.id.showAnswerButton);
        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
            }
        });
    }
}
```

Код достаточно тривиален: мы задаем текст `TextView` при помощи метода `TextView.setText(int)`. Метод `TextView.setText(…)` существует в нескольких вариантах; здесь используется вариант, получающий идентификатор строкового ресурса.

Запустите приложение GeoQuiz. Нажмите кнопку Cheat!, чтобы перейти к `CheatActivity`. Затем нажмите кнопку Show Answer, чтобы открыть ответ на текущий вопрос.

Получение результата от дочерней активности

В текущей версии приложения пользователь может беспрепятственно жульничать. Сейчас мы исправим этот недостаток; для этого `CheatActivity` будет сообщать `QuizActivity`, подсмотрел ли пользователь ответ.

Чтобы получить информацию от дочерней активности, вызовите следующий метод `Activity`:

```
public void startActivityForResult(Intent intent, int requestCode)
```

Первый параметр содержит тот же интент, что и прежде. Во втором параметре содержится *код запроса* — определяемое пользователем целое число, которое передается дочерней активности, а затем принимается обратно родителем. Оно используется тогда, когда активность запускает сразу несколько типов дочерних активностей и ей необходимо определить, кто из потомков возвращает данные.

В классе QuizActivity измените слушателя mCheatButton и включите в него вызов startActivityForResult(Intent, int).

Листинг 5.13. Вызов startActivityForResult(...) (QuizActivity.java)

```
        ...
        mCheatButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new Intent(QuizActivity.this, CheatActivity.class);
                boolean answerIsTrue = mAnswerKey[mCurrentIndex].isTrueQuestion();
                i.putExtra(CheatActivity.EXTRA_ANSWER_IS_TRUE, answerIsTrue);
                startActivity(i);
                startActivityForResult(i, 0);
            }
        });

        updateQuestion();
    }
```

QuizActivity всегда запускает дочерние активности только одного типа. Значение второго обязательного параметра в этом случае несущественно, поэтому мы будем передавать 0.

Передача результата

Существует два метода, которые могут вызываться в дочерней активности для возвращения данных родителю:

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

Как правило, resultCode содержит одну из двух предопределенных констант: Activity.RESULT_OK или Activity.RESULT_CANCELED. (Также можно использовать другую константу RESULT_FIRST_USER как смещение при определении собственных кодов результатов.)

Назначение кода результата полезно в том случае, когда родитель должен выполнить разные действия в зависимости от того, как завершилась дочерняя активность.

Например, если в дочерней активности присутствуют кнопки OK и Cancel, то дочерняя активность может назначать разные коды результата в зависимости от того, какая кнопка была нажата. Родительская активность будет выбирать разные варианты действий в зависимости от полученного кода.

Вызов setResult(…) не является обязательным для дочерней активности. Если вам не нужно различать результаты или получать произвольные данные по интенту, просто разрешите ОС отправить код результата по умолчанию. Код результата всегда возвращается родителю, если дочерняя активность была запущена методом startActivityForResult(…). Если метод setResult(…) не вызывался, то при нажатии пользователем кнопки Back родитель получит код Activity.RESULT_CANCELED.

Возвращение интента

В нашей реализации дочерняя активность должна вернуть QuizActivity некоторые данные. Соответственно, мы создадим объект Intent, поместим в него дополнение, а затем вызовем Activity.setResult(int, Intent) для передачи этих данных QuizActivity.

Ранее в приложении была определена константа для дополнения, получаемого CheatActivity внутри CheatActivity. То же самое следует сделать для нового дополнения, которое CheatActivity отправляет QuizActivity. Это объясняется тем, что входящие и исходящие дополнения определяют интерфейс CheatActivity. Если вы захотите использовать CheatActivity в другом месте приложения, вы будете ссылаться на константы, определенные исключительно в CheatActivity.

Добавьте в CheatActivity константу для ключа дополнения и закрытый метод, который создает интент, помещает в него дополнение и назначает код результата. Затем включите вызов этого метода в слушателя кнопки Show Answer.

Листинг 5.14. Назначение результата (CheatActivity.java)

```
public class CheatActivity extends Activity {

    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    public static final String EXTRA_ANSWER_SHOWN =
        "com.bignerdranch.android.geoquiz.answer_shown";
    ...
    private void setAnswerShownResult(boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);
        setResult(RESULT_OK, data);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        // Ответ будет показан только в том случае,
        // если пользователь нажмет кнопку
        setAnswerShownResult(false);
        ...
        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
                setAnswerShownResult(true);
            }
        });
    }
```

Когда пользователь нажимает кнопку Show Answer, `CheatActivity` упаковывает код результата и интент в вызове `setResult(int, Intent)`.

Затем, когда пользователь нажимает кнопку Back для возвращения к `QuizActivity`, `ActivityManager` вызывает следующий метод родительской активности:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

В параметрах передается исходный код запроса от `QuizActivity`, код результата и интент, переданный `setResult(…)`.

Последовательность взаимодействий изображена на рис. 5.12.

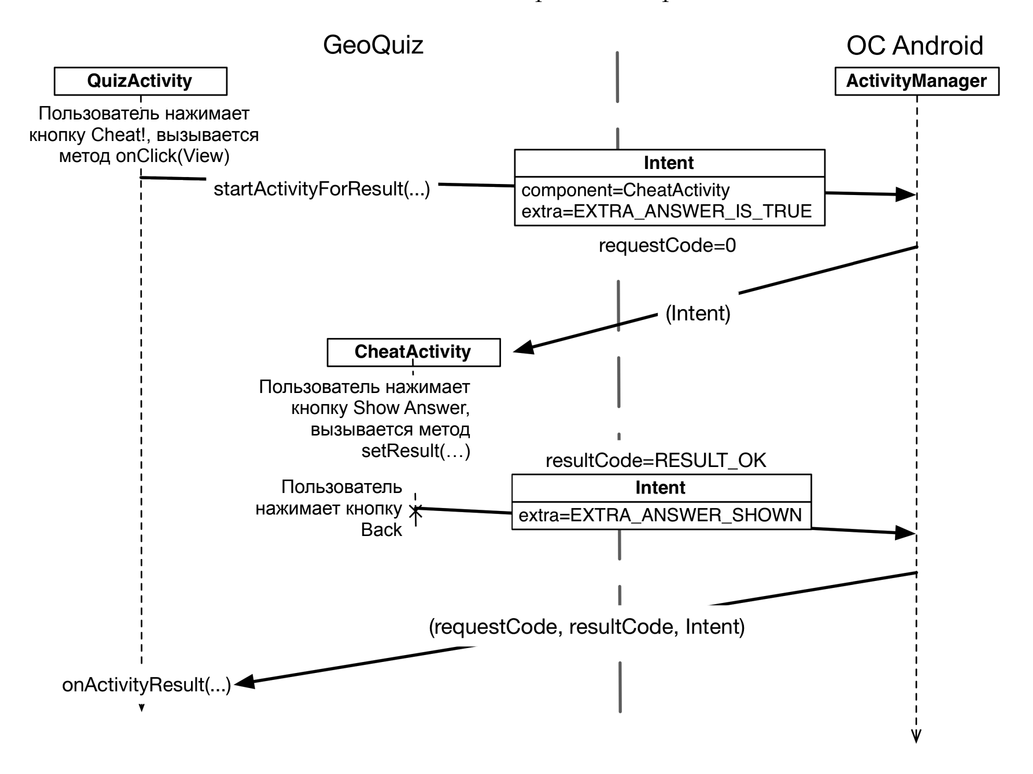


Рис. 5.12. Диаграмма последовательности для GeoQuiz

Остается последний шаг: переопределение `onActivityResult(int, int, Intent)` в `QuizActivity` для обработки результата.

Обработка результата

Добавьте в QuizActivity.java новую переменную для хранения значения, возвращаемого `CheatActivity`. Затем включите в переопределение `onActivityResult(…)` код его получения.

Листинг 5.15. Реализация onactivityResult(...) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private int mCurrentIndex = 0;

    private boolean mIsCheater;
    ...

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (data == null) {
          return;
        }
        mIsCheater = data.getBooleanExtra(CheatActivity.EXTRA_ANSWER_SHOWN, false);
    }
    ...
}
```

В этой реализации `onActivityResult(…)` `QuizActivity` не проверяет ни код запроса, ни код результата. В других ситуациях в программу могут включаться условные конструкции, основанные на проверке этих значений.

Наконец, измените метод `checkAnswer(boolean)` в `QuizActivity`. Он должен проверять, подсмотрел ли пользователь ответ, и реагировать соответствующим образом.

Листинг 5.16. Изменение уведомления в зависимости от значения mIsCheater (QuizActivity.java)

```
    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mAnswerKey[mCurrentIndex].isTrueQuestion();

        int messageResId = 0;

        if (mIsCheater) {
            messageResId = R.string.judgment_toast;
        } else {
            if (userPressedTrue == answerIsTrue) {
                messageResId = R.string.correct_toast;
            } else {
                messageResId = R.string.incorrect_toast;
            }
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mNextButton = (Button)findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mAnswerKey.length;
```

```
                mIsCheater = false;
                updateQuestion();
            }
        });
        ...
    }
```

Запустите приложение GeoQuiz. Попробуйте подсмотреть ответ и посмотрите, что произойдет.

Ваши активности с точки зрения Android

Давайте посмотрим, что происходит при переключении между активностями с точки зрения ОС. Прежде всего, когда вы щелкаете на приложении GeoQuiz в лаунчере, ОС запускает не приложение, а активность в приложении. А если говорить точнее, запускается *активность лаунчера* приложения. Для GeoQuiz активностью лаунчера является `QuizActivity`.

Когда шаблон создавал приложение GeoQuiz, класс `QuizActivity` был назначен активностью лаунчера по умолчанию. Статус активности лаунчера задается в манифесте элементом `intent-filter` в объявлении `QuizActivity` (листинг 5.17).

Листинг 5.17. QuizActivity объявляется активностью лаунчера (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  ...
  <application
    ... >
    <activity
      android:name="com.bignerdranch.android.geoquiz.QuizActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".CheatActivity"
      android:label="@string/app_name" />
  </application>

</manifest>
```

Когда экземпляр `QuizActivity` окажется на экране, пользователь может нажать кнопку Cheat!. При этом экземпляр `CheatActivity` запускается поверх `QuizActivity`. Эти активности образуют стек (рис. 5.13).

При нажатии кнопки Back в `CheatActivity` этот экземпляр выводится из стека, а `QuizActivity` занимает свою позицию на вершине.

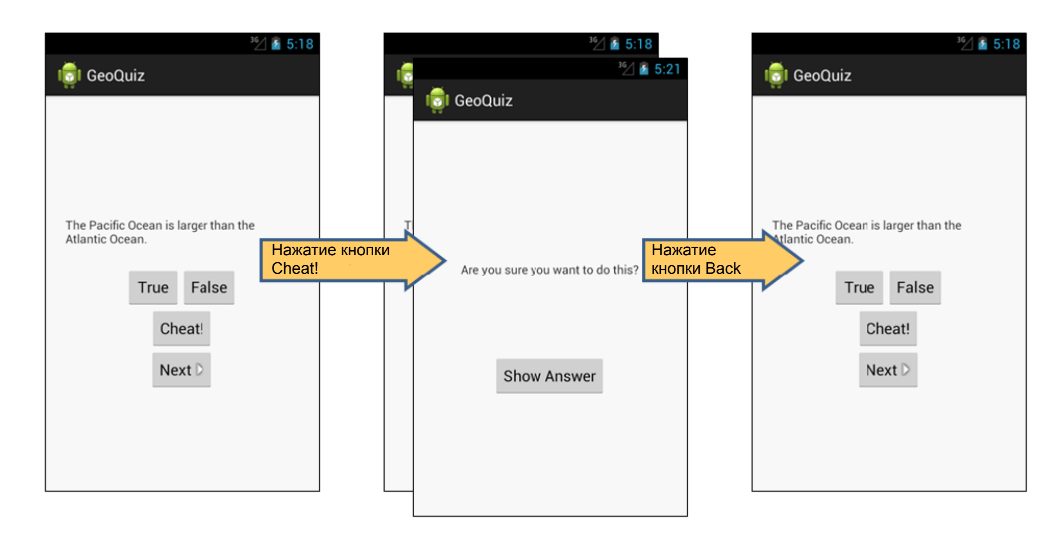


Рис. 5.13. Стек GeoQuiz

Вызов `Activity.finish()` в `CheatActivity` также выводит `CheatActivity` из стека. Если запустить GeoQuiz из Eclipse и нажать кнопку Back в QuizActivity, то активность `QuizActivity` будет извлечена из стека и вы вернетесь к последнему экрану, который просматривался перед запуском GeoQuiz.

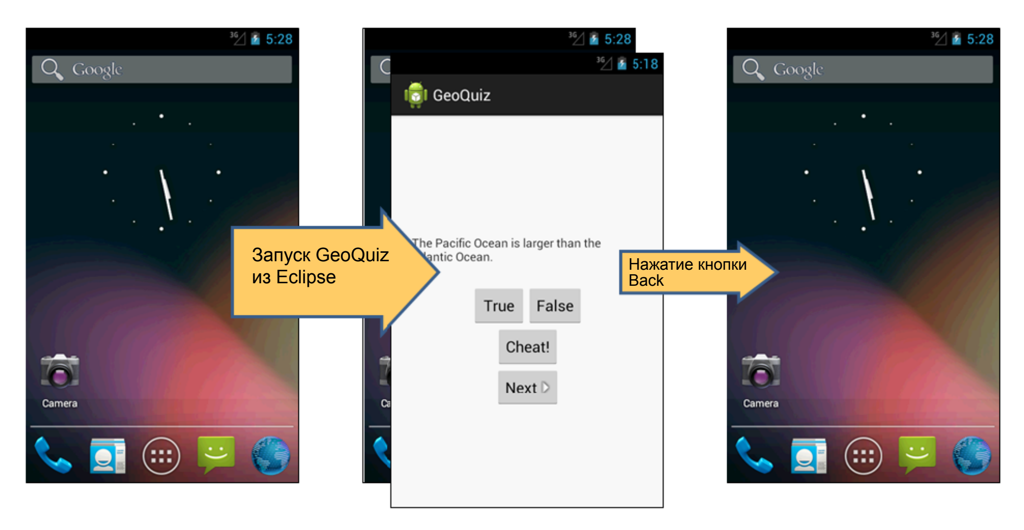


Рис. 5.14. Экран Home при запуске из Eclipse

Если вы запустили GeoQuiz из лаунчера (launcher), то нажатие кнопки Back из `QuizActivity` вернет вас обратно.



Рис. 5.15. Запуск GeoQuiz из лаунчера

Нажатие Back в запущенном лаунчере вернет вас к экрану, который был открыт перед его запуском.

Мы видим, что `ActivityManager` поддерживает стек возврата (back stack) и что этот стек не ограничивается активностями вашего приложения. Он совместно используется активностями всех приложений; это одна из причин, по которым `ActivityManager` участвует в запуске активностей и находится под управлением ОС, а не вашего приложения. Стек представляет использование ОС и устройства в целом, а не использование одного приложения.

Упражнение

Мошенники никогда не выигрывают... Если, конечно, им не удастся обойти вашу защиту от мошенничества. А скорее всего, они так и сделают — именно потому, что они мошенники.

Приложение GeoQuiz содержит ряд «лазеек», которыми могут воспользоваться мошенники. В этом упражнении вы должны устранить эти лазейки. Основные дефекты приложения перечислены ниже по возрастанию сложности, от самых простых к самым сложным:

- Подсмотрев ответ, пользователь может повернуть `CheatActivity`, чтобы сбросить результат.
- После возвращения пользователь может повернуть `QuizActivity`, чтобы сбросить флаг `mIsCheater`.
- Пользователь может нажимать кнопку `Next` до тех пор, пока вопрос, ответ на который был подсмотрен, снова не появится на экране.

Удачи!

6 Версии Android SDK и совместимость

Теперь, после «боевого крещения» с приложением GeoQuiz, пора поближе познакомиться с разными версиями Android. Информация этой главы пригодится вам в следующих главах книги, когда мы займемся разработкой более сложных и реалистичных приложений.

Версии Android SDK

В таблице 6.1 перечислены версии SDK, соответствующие версии прошивки Android, и процент устройств, использующих их, по состоянию на март 2013 года.

Таблица 6.1. Уровни API Android, версии прошивки и процент устройств

Уровень API	Кодовое название	Версия прошивки устройства	% использующих устройств
17	Jelly Bean	4.2	1,6
16		4.1	14,9
15	Ice Cream Sandwich (ICS)	4.0.3, 4.0.4	28,6
13	Honeycomb (только для планшетов)	3.2	0,9
12		3.1.x	0,3
10	Gingerbread	2.3.3–2.3.7	43,9
9		2.3.3, 2.3.1, 2.3	0,2
8	Froyo	2.2.x	7,5
7	Eclair	2.1.x	1,9

За каждым выпуском с кодовым названием следуют инкрементные выпуски. Например, платформа Ice Cream Sandwich was изначально выпущена как Android 4.0 (API уровня 14) и почти немедленно заменена инкрементными выпусками, которые в конечном итоге привели к появлению Android 4.0.3 и 4.0.4 (API уровня 15).

Конечно, проценты из табл. 6.1 будут изменяться, но в них проявляется важная закономерность: устройства Android со старыми версиями не подвергаются немедленному обновлению или замене при появлении новой версии. По состоянию на март 2013 года на половине устройств все еще работали версии Froyo или Gingerbread SDK. Версия Android 2.3.7 (последнее обновление Gingerbread) была выпущена в сентябре 2011 года. С другой стороны, версия Android 4.2, выпущенная в ноябре 2012, работает только на 1,6 % устройств.

(Обновленные данные из таблицы доступны по адресу *http://developer.android.com/resources/dashboard/platform-versions.html*.)

Почему на многих устройствах продолжают работать старые версии Android? В основном из-за острой конкуренции между производителями устройств Android и операторами сотовой связи. Операторы стремятся иметь возможности и телефоны, которых нет у других сетей. Производители устройств тоже испытывают давление — все их телефоны базируются на одной ОС, но им нужно выделяться на фоне конкурентов. Сочетание этого давления со стороны рынка и операторов сотовой связи привела к появлению многочисленных устройств со специализированными модификациями Android.

Устройство со специализированной версией Android не сможет перейти на новую версию, выпущенную Google. Вместо этого ему придется ждать совместимого «фирменного» обновления, которое может выйти через несколько месяцев после выпуска версии Google... А может вообще не выйти — производители часто предпочитают расходовать ресурсы на новые устройства, а не на обновление старых устройств. Иногда оборудованию старого устройства попросту не хватает производительности для запуска новой версии Android.

Совместимость и программирование Android

Из-за задержек обновления в сочетании с регулярным выпуском новых версий совместимость становится важной проблемой в программировании Android. Чтобы привлечь широкую аудиторию, разработчики Android должны создавать приложения, которые хорошо работают на устройствах с разными версиями Android: Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich и Jelly Bean, а также на устройствах различных форм-факторов.

Поддержка разных размеров не столь сложна, как может показаться. Экраны телефонов существуют во многих вариантах размеров, но система макетов Android хорошо приспосабливается к ним. С планшетами дело обстоит сложнее, но в этом случае на помощь приходят квалификаторы конфигураций (как будет показано в главе 22). Впрочем, для Google TV (также работающим под управлением Android) различия в пользовательском интерфейсе настолько серьезны, что обычно приходится использовать отдельное приложение.

Версии — совсем другое дело. С инкрементными выпусками обратная совместимость обычно не создает проблем, однако в истории был один особенно принципиальный сдвиг.

Трудности с Honeycomb

Самые большие проблемы с совместимостью Android связаны с преодолением границы между «мирами» до и после Honeycomb. Выпуск Honeycomb стал поворотным моментом в Android, ознаменовавшим появление нового пользовательского интерфейса и новых архитектурных компонентов. Версия Honeycomb предназначалась только для планшетов и Google TV, количество ее пользователей было ограничено, так что новые разработки получили массовое распространение только с выходом Ice Cream Sandwich. Версии, выходившие после этого, были менее революционными.

Учитывая, что больше половины устройств все еще работает под управлением Gingerbread или более старых версий, разработчики не могут начать «с чистого листа» и отказаться от того, что было раньше. Со временем доля старых устройств сокращается, но это происходит намного медленнее, чем можно представить.

Следовательно, разработчик Android должен выделить время на обеспечение обратной совместимости и преодоление разрыва между Gingerbread (API уровня 10) и Honeycomb (API уровня 11). Android предоставляет поддержку для поддержания обратной совместимости; также существуют сторонние библиотеки, которые помогают в решении этой задачи. Тем не менее поддержание совместимости усложняет изучение программирования Android.

Нередко разработчику приходится изучать два способа выполнения некоторой операции, а также учиться объединять их. В других случаях способ всего один, но он выглядит чрезмерно усложненным, потому что ориентируется на два (как минимум) набора требований.

Мы рекомендуем погрузиться в сон на пару лет (если получится, конечно). Потом вы проснетесь и начнете изучать программирование Android, когда устройства Gingerbread уже не будут составлять значимую долю рынка. А если это невозможно, постарайтесь хотя бы понять причины некоторых сложностей.

При создании проекта GeoQuiz в мастере нового приложения выбираются три версии SDK. (Учтите, что в Android термины «версия SDK» и «уровень API» являются синонимами.)

Давайте посмотрим, какое место в проекте занимает каждая из этих настроек, изучим значения по умолчанию и способы их изменения.

Минимальная необходимая версия SDK (Minimum Required SDK) и целевая версия SDK (Target SDK) задаются в манифесте. Еще раз откройте AndroidManifest.xml из Package Explorer. В элементе `uses-sdk` найдите значения `android:minSdkVersion` и `android:targetSdkVersion`.

Листинг 6.1. minSdkVersion в манифесте (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.geoquiz"
```

```
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
  ...

</manifest>
```

Рис. 6.1. Еще не забыли?

Минимальная версия SDK (Minimum Required SDK)

Вспомните, что манифест содержит метаданные, используемые ОС для взаимодействия с вашим приложением. Значение `minSdkVersion` определяет нижнюю границу, за которой ОС отказывается устанавливать приложение.

Выбирая API уровня 8 (Froyo), вы разрешаете Android устанавливать GeoQuiz на устройствах с версией Froyo и выше. Android откажется устанавливать GeoQuiz на устройстве, допустим, с системой Eclair.

Снова обращаясь к табл. 6.1, мы видим, почему Froyo является хорошим выбором для минимальной версии SDK: в этом случае приложение можно будет устанавливать на 95 % используемых устройств.

Целевая версия SDK (Target SDK)

Значение `targetSdkVersion` сообщает Android, для какого уровня API проектировалось ваше приложение. Чаще всего в этом поле указывается новейшая версия Android.

Когда следует понижать целевую версию SDK? Новые выпуски SDK могут изменять внешний вид вашего приложения на устройстве и даже поведение ОС «за кулисами». Если ваше приложение уже спроектировано, убедитесь в том, что в новых версиях оно работает так, как ожидалось. За информацией о возможных проблемах обращайтесь к документации по адресу *http://developer.android.com/reference/android/os/Build.VERSION_CODES.html*. Далее либо измените свое приложение, чтобы оно работало с новым поведением, либо понизьте целевую версию SDK. Понижение целевой версии SDK гарантирует, что приложение будет работать с внешним видом и поведением целевой версии, в которой оно хорошо работало. Все изменения последующих версий при этом игнорируются.

Версия SDK для построения (Compile With)

В последнем поле для выбора SDK (CompileWith на рис. 6.1) вводится версия SDK для построения. В манифесте этот параметр отсутствует. Если минимальная и целевая версии SDK сообщаются ОС, версия SDK для построения относится к закрытой информации, известной только вам и компилятору.

Функциональность Android используется через классы и методы SDK. Версия SDK, используемая для построения, также называемая *целью построения* (build target), указывает, какая версия должна использоваться при построении вашего кода. Когда Eclipse ищет классы и методы, на которые вы ссылаетесь в директивах импорта, версия SDK для построения определяет, в какой версии SDK будет осуществляться поиск.

Чтобы сменить цель построения, щелкните правой кнопкой мыши на проекте GeoQuiz на панели Package Explorer и выберите команду Properties. В левой части диалогового окна выберите категорию Android; открывается полный список вариантов.

Чем отличаются цели построения Google APIs и Android Open Source Project? Google APIs включает Android API и дополнения Google — прежде всего API для работы с Google Maps.

Цель построения GeoQuiz выбрана нормально. Щелкните на кнопке Cancel, будем двигаться дальше.

Безопасное добавление кода для более поздних версий API

Различия между минимальной версией SDK и версией SDK для построения создают проблемы совместимости, которыми необходимо управлять. Например, что произойдет в GeoQuiz при выполнении кода, рассчитанного на версию SDK после минимальной версии Froyo (API уровня 8)? Если установить такое приложение и запустить его на устройстве Froyo, произойдет сбой.

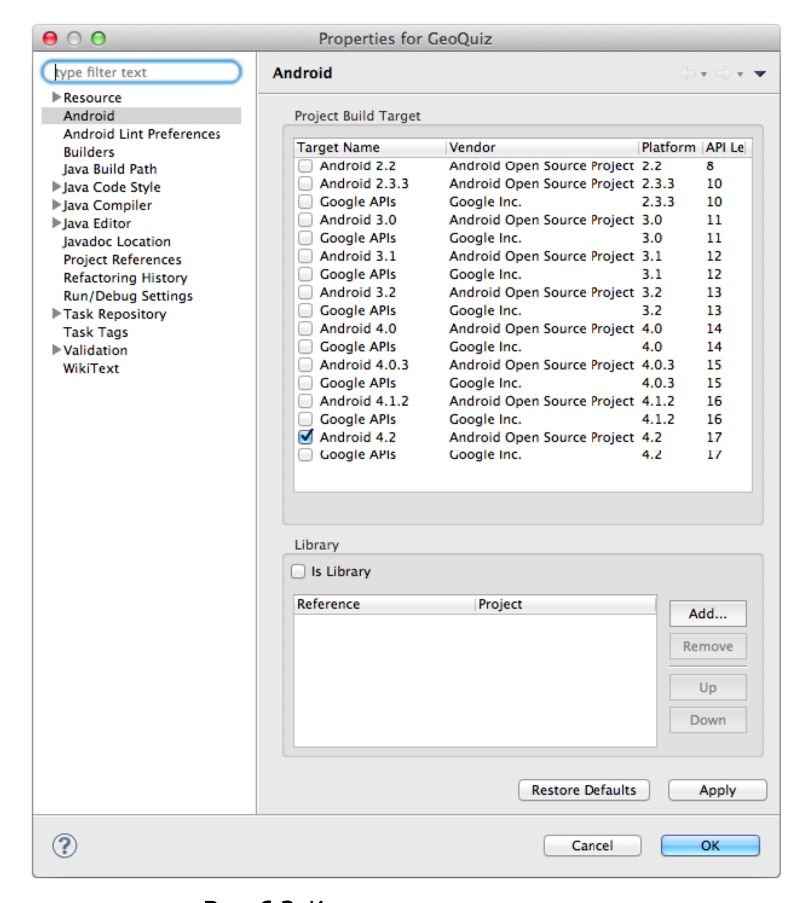


Рис. 6.2. Изменение цели построения

Раньше тестирование в подобных ситуациях было сущим кошмаром. Однако благодаря совершенствованию Android Lint проблемы, вызванные вызовом нового кода на старых устройствах, теперь успешно обнаруживаются. Если вы используете код версии, превышающей минимальную версию SDK, Android Lint сообщит об ошибке построения.

Весь код текущей версии GeoQuiz совместим с API уровня 8 и ниже. Добавим код API уровня 11 и посмотрим, что произойдет.

Откройте файл QuizActivity.java. Включите в метод `onCreate(Bundle)` следующий фрагмент для создания подзаголовка с указанием географической области, к которой относится вопрос.

Листинг 6.2. Добавление подзаголовка (QuizActivity.java)

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);
```

продолжение ⇗

Листинг 6.2 (*продолжение*)

```
ActionBar actionBar = getActionBar();
actionBar.setSubtitle("Bodies of Water");

    mIsCheater = false;
```

Строковый литерал используется для простоты. Если бы нам действительно потребовалось выводить подзаголовок (или даже разные подзаголовки для разных групп вопросов), то следовало бы добавить в программу строковые ресурсы и использовать ссылки на них в коде.

Проведите организацию импорта, чтобы импортировать класс `ActionBar`. Этот класс появился в API уровня 11, поэтому при попытке выполнения кода на устройстве с более ранней версией произойдет сбой. Мы еще вернемся к классу `ActionBar` в главе 16, а пока просто используем его как пример кода, не поддерживаемого Froyo.

После организации импорта выберите GeoQuiz на панели Package Explorer и выполните команду Android Tools ▸ Run Lint: Check for Common Errors. Так как для построения используется API уровня 17, у компилятора не возникнет проблем с этим кодом. С другой стороны, Android Lint знает минимальную версию SDK и будет протестовать.

Сообщения об ошибках будут выглядеть примерно так: «Class requires API level 11 (current min is8)». Android Lint не позволит построить приложение до тех пор, пока не будет решена проблема совместимости.

Как избавиться от ошибок? Первый способ — поднять минимальную версию SDK до 11. Однако тем самым вы не столько решаете проблему совместимости, сколько обходите ее. Если ваше приложение не может устанавливаться на устройствах Gingerbread и более старых устройствах, то проблема исчезает.

Другое, более правильное решение — заключить код `ActionBar` в условную конструкцию, которая проверяет версию Android на устройстве.

Листинг 6.3. Предварительная проверка версии Android на устройстве

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setSubtitle("Bodies of Water");
    }
}
```

Константа `Build.VERSION.SDK_INT` определяет версию Android на устройстве. Она сравнивается с константой, соответствующей Honeycomb. (Коды версий доступны по адресу *http://developer.android.com/reference/android/os/Build.VERSION_CODES.html*.)

Теперь код `ActionBar` будет выполняться только в том случае, если приложение работает на устройстве с версией Honeycomb и выше. Код стал безопасным для

Froyo, вроде бы у Android Lint не на что жаловаться. Однако попытавшись запустить приложение снова, вы получите ту же ошибку.

Подавление ошибок совместимости Lint

К сожалению, Android Lint не может понять, что вы сделали, поэтому выдачу ошибки придется подавить явно. Включите следующую аннотацию перед реализацией `onCreate(Bundle)`.

Листинг 6.4. Полезная информация для Android Lint

```
@TargetApi(11)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setSubtitle("TFFTT");
    }
}
```

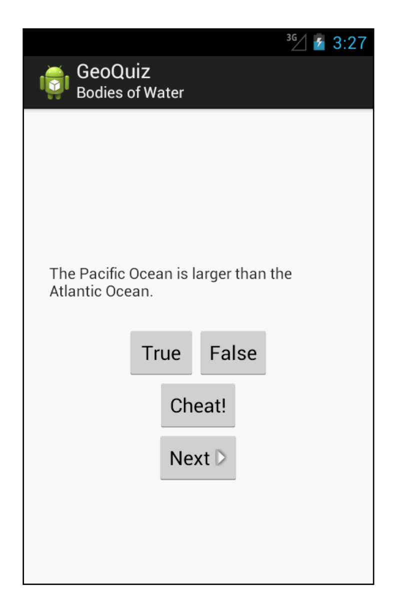


Рис. 6.3. Приложение с подзаголовком

Чтобы понять, почему команда `if` должна дополняться аннотацией, представьте программирование Android в виде пляжа. В воде плавают акулы — исключения времени выполнения от использования новых методов или классов на старых устройствах. Android Lint — спасатель на этом пляже. Он следит за обстановкой, всегда готовый прыгнуть вводу и спасти вас, если поблизости окажется акула.

Код в листинге 6.4 использует химический репеллент для акул и отмахивается от спасателя. Команда `if` — это ваш репеллент: метод `getActionBar()`, заключенный в конструкцию `if`, вызывается только в случае его доступности, и акулы на вас не нападут. Аннотация `@TargetApi(11)` как бы отмахивается от спасателя (Android Lint), говоря ему: «Не обращай внимания на акул — у меня все под контролем». Спасатель решает, что вам виднее, и не пытается вытаскивать вас из воды.

Итак, если вы отмахиваетесь от спасателя аннотацией `@TargetApi`, пожалуйста, не забудьте про свой репеллент `SDK_INT`. Если вы этого не сделаете, вас съедят акулы исключений.

Запустите GeoQuiz на устройстве с версией Honeycomb и выше и проверьте новый подзаголовок.

Также можно запустить GeoQuiz на устройстве Froyo или Gingerbread (виртуальном или физическом). Подзаголовок в этом случае не отображается, но вы можете убедиться в том, что приложение работает нормально.

Документация разработчика Android

Ошибки Android Lint сообщают, к какому уровню API относится несовместимый код. Однако вы также можете узнать, к какому уровню API относятся конкретные классы и методы, — эта информация содержится в документации разработчика Android.

Постарайтесь поскорее научиться работать с документацией. Информация Android SDK слишком обширна, чтобы держать ее в голове, а с учетом регулярного появления новых версий разработчик должен знать, что нового появилось, и уметь пользоваться этими новшествами.

Документация разработчика Android — превосходный и объемный источник информации. Ее главная страница находится по адресу *http://developer.android.com/*. Документация делится на три части: проектирование (Design), разработка (Develop) и распространение (Distribute). В разделе проектирования описаны паттерны и принципы проектирования пользовательского интерфейса приложений. В разделе разработки содержится основная документация и учебные материалы. В разделе распространения объясняется, как готовить и публиковать приложения в Google Play или через механизм открытого распространения. Все это стоит просмотреть, когда у вас появится возможность.

Раздел разработки дополнительно разбит на четыре секции.

Android Training	Учебные модули для начинающих и опытных разработчиков, включая загружаемый код примеров
API Guides	Тематические описания компонентов приложений, функций и полезных приемов
Reference	Гипертекстовая документация по всем классам, методам, интерфейсам, константам атрибутов и т. д. в SDK, с возможностью поиска
Tools	Описания и ссылки на инструменты разработчика

Для работы с документацией не обязательно иметь доступ к Интернету. Откройте в файловой системе каталог, в который были загружены SDK; в нем расположен каталог docs с полной документацией.

Чтобы определить, к какому уровню API относится `getActionBar()`, проведите поиск этого метода при помощи строки поиска в правом верхнем углу браузера. Первым результатом будет руководство по API с описанием строки заголовка.

Однако нам нужны результаты из справочной секции. Отфильтруйте результаты поиска, щелкнув на категории Reference в левой части окна.

Выберите первый результат; открывается справочная страница с описанием класса Activity. В верхней части страницы находятся ссылки на разные секции. Щелкните на ссылке Methods, чтобы получить список методов `Activity`.

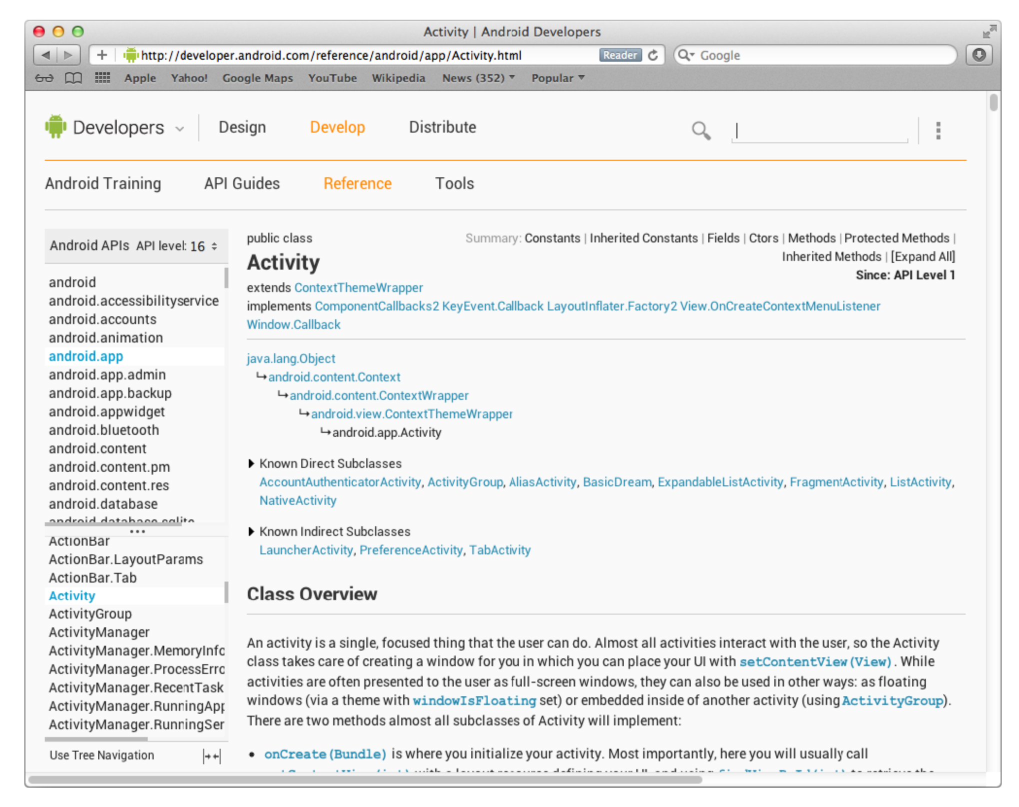


Рис. 6.4. Страница со справочным описанием Activity

Прокрутите список, найдите метод `getActionBar()` и щелкните на имени метода, чтобы увидеть описание. Справа от сигнатуры метода видно, что метод `getAction-Bar()` появился в API уровня 11.

Если вы хотите узнать, какие методы `Activity` доступны, скажем, в API уровня 8, отфильтруйте справочник по уровню API. В левой части страницы, где классы

индексируются по пакету, найдите категорию API level:17. Щелкните на близлежащем элементе управления и выберите в списке 8. Все, что появилось в Android после API уровня 8, выделяется в списке серым цветом.

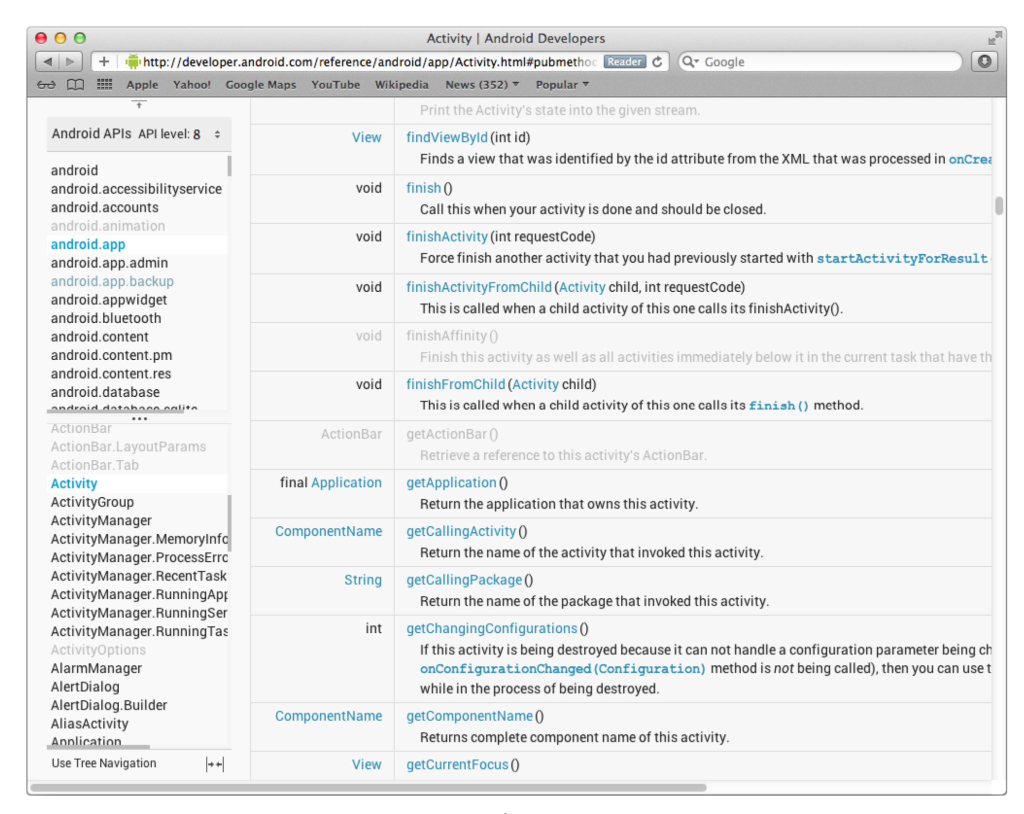


Рис. 6.5. Методы Activity, отфильтрованные по API уровня 8

Продолжая чтение книги, постарайтесь часто возвращаться к документации. Она наверняка понадобится вам для выполнения упражнений, однако не стесняйтесь заниматься самостоятельными исследованиями каждый раз, когда вам захочется побольше узнать о некотором классе, методе и т. д. Android постоянно обновляет и совершенствует свою документацию, и вы всегда узнаете из нее что-то новое.

Упражнение. Вывод версии построения

Добавьте в макет GeoQuiz виджет `TextView` для вывода уровня API устройства, на котором работает программа.

Задать текст `TextView` в макете невозможно, потому что версия построения устройства неизвестна до момента выполнения. Найдите метод `TextView` для задания текста в справочной странице `TextView` в документации Android. Вам нужен метод, получающий один аргумент — строку (или `CharSequence`).

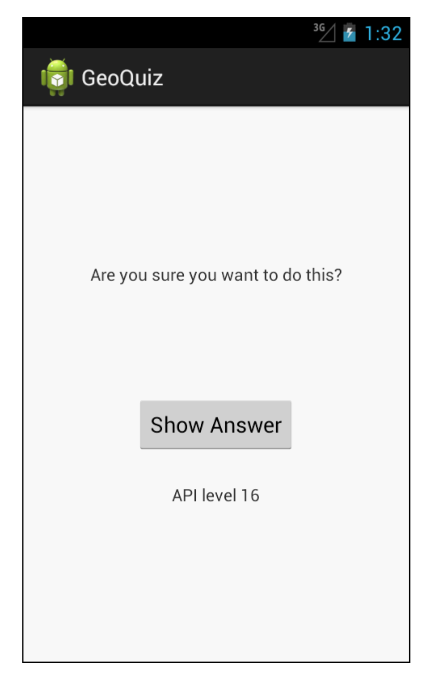


Рис. 6.6. Результат упражнения

Для настройки размера и гарнитуры текста используйте атрибуты XML, перечисленные в описании `TextView`.

7 UI-фрагменты и FragmentManager

В этой главе мы начнем строить приложение CriminalIntent. Оно предназначено для хранения информации об «офисных преступлениях»: грязной посуде, оставленной в раковине, или пустом лотке общего принтера после печати документов.

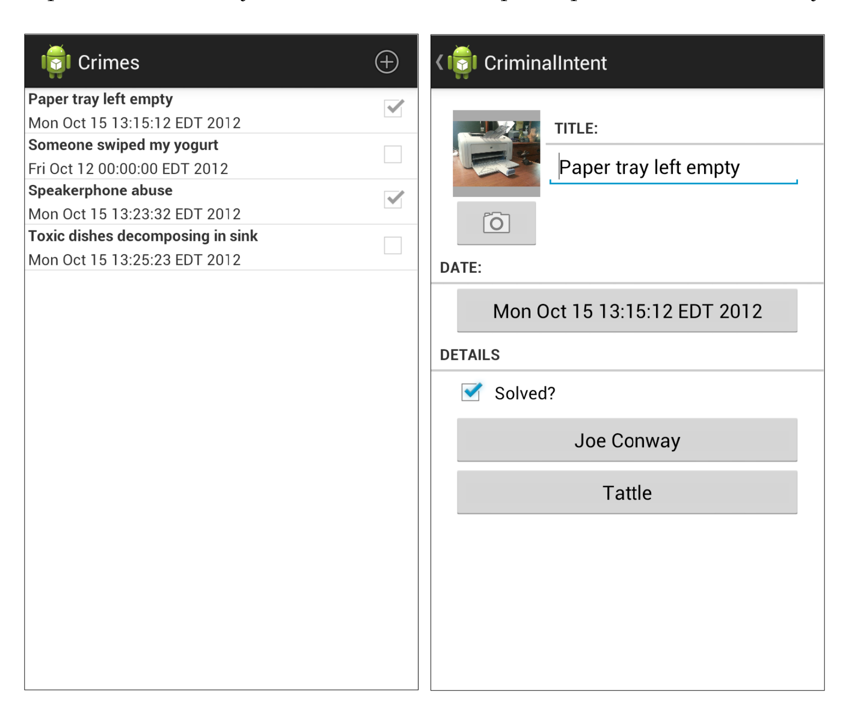


Рис. 7.1. Приложение CriminalIntent

В приложении CriminalIntent пользователь создает запись о преступлении с заголовком, датой и фотографией. Также можно выбрать подозреваемого в списке контактов и отправить жалобу по электронной почте, опубликовать ее в Twitter, Facebook или другом приложении. Сообщив о преступлении, пользователь освобождается от негатива и может сосредоточиться на текущей задаче.

CriminalIntent — сложное приложение, для построения которого нам понадобится целых 13 глав. В нем используется интерфейс типа «список/детализация»: на главном экране выводится список зарегистрированных преступлений. Пользователь может добавить новое или выбрать существующее преступление для просмотра и редактирования информации.

Гибкость пользовательского интерфейса

Разумно предположить, что приложение типа «список/детализация» состоит из двух активностей: для управления списком и для управления детализированным представлением. Щелчок на преступлении в списке запускает экземпляр детализированной активности.

Нажатие кнопки Back уничтожает активность детализации и возвращает на экран список, в котором можно выбрать другое преступление.

Такая архитектура работает, — но что делать, если вам потребуется более сложная схема представления информации и навигации между экранами?

- Допустим, пользователь запустил приложение CriminalIntent на планшете. Экраны планшетов и некоторых больших телефонов достаточно велики для одновременного отображения списка и детализации — по крайней мере в альбомной ориентации.

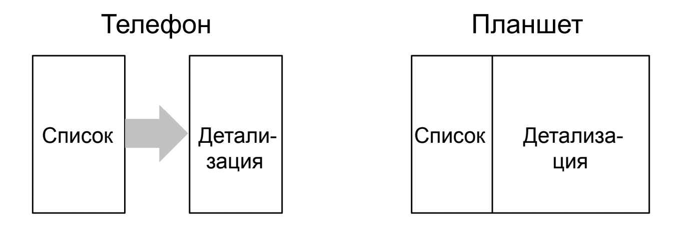


Рис. 7.2. Идеальный интерфейс «список/детализация» для телефонов и планшетов

- Пользователь просматривает описание преступления на телефоне и хочет увидеть следующее преступление в списке. Было бы удобно, если бы пользователь мог провести пальцем по экрану, чтобы перейти к следующему преступлению без возвращения к списку. Каждый жест прокрутки должен обновлять детализированное представление информацией о следующем преступлении.

Эти сценарии объединяет гибкость пользовательского интерфейса: возможность формирования и изменения представления активности во время выполнения в зависимости от того, что нужно пользователю или устройству.

Подобная гибкость в активности не предусмотрена. Активность тесно связана со своим представлением. Представление, которое заполняется активностью при вызове `setContentView(…)`, привязывается к активности до самого конца. Вы не можете переключить все представление активности (не уничтожая саму активность) или передать представление от одной активности к другой во время выполнения. Таков закон Android.

Знакомство с фрагментами

Закон Android нельзя нарушить, но можно обойти, передав управление пользовательским интерфейсом приложения от активности одному или нескольким *фрагментам* (fragments).

Фрагмент представляет собой объект контроллера, которому активность может доверить выполнение операций. Чаще всего такой операцией является управление пользовательским интерфейсом — целым экраном или его частью.

Фрагмент, управляющий пользовательским интерфейсом, называется *UI-фрагментом*. UI-фрагмент имеет собственное представление, которое заполняется на основании файла макета. Представление фрагмента содержит элементы пользовательского интерфейса, с которыми будет взаимодействовать пользователь.

Пользователь активности содержит точку, в которой вставляется представление фрагмента, или несколько точек для представлений нескольких фрагментов.

Фрагменты, связанные с активностью, могут использоваться для формирования и изменения экрана в соответствии с потребностями приложения и пользователей. Представление активности формально остается неизменным на протяжении жизненного цикла, и законы Android не нарушаются.

Давайте посмотрим, как эта схема работает в приложении «список/детализация». Представление активности строится из фрагмента списка и фрагмента детализации. Представление детализации содержит подробную информацию о выбранном элементе списка.

При выборе другого элемента на экране появляется новое детализированное представление. С фрагментами эта задача решается легко; активность заменяет фрагмент детализации другим фрагментом детализации (рис. 7.3). Это существенное изменение представления происходит без уничтожения активности.

Применение UI-фрагментов позволяет разделить пользовательский интерфейс вашего приложения на структурные блоки, а это полезно не только для приложений «список/детализация». Работа с отдельными блоками упрощает построение интерфейсов со вкладками, анимированных боковых панелей и многих других.

Впрочем, за такую гибкость пользовательского интерфейса приходится платить: повышением сложности, увеличением количества «подвижных частей», увеличением объема кода.

Выгода из использования фрагментов проявится в главах 11 и 22, зато разбираться со сложностью придется прямо сейчас.

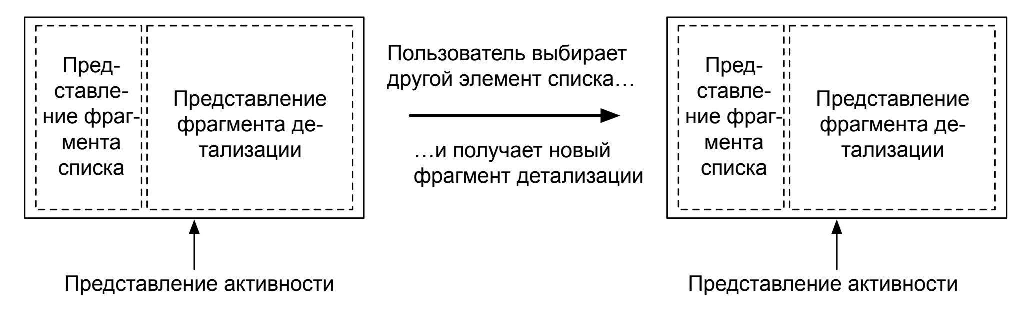


Рис. 7.3. Переключение фрагмента детализации

Начало работы над CriminalIntent

В этой главе мы возьмемся за представление детализации CriminalIntent. На рис. 7.4 показано, как будет выглядеть приложение CriminalIntent к концу главы.

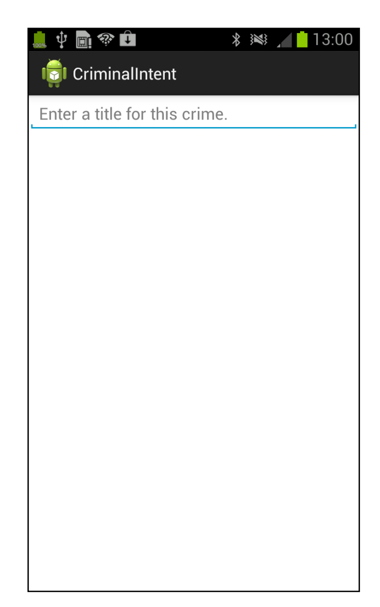


Рис. 7.4. Приложение CriminalIntent к концу главы

На первый взгляд результат не впечатляет. Однако не забывайте, что эта глава всего лишь закладывает основу для более серьезных дел в будущем.

Экраном, показанным на рис. 7.4, будет управлять UI-фрагмент с именем `CrimeFragment`. *Хостом* (host) экземпляра `CrimeFragment` является активность с именем `CrimeActivity`.

Пока считайте, что хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рис. 7.5). Фрагмент не может

вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

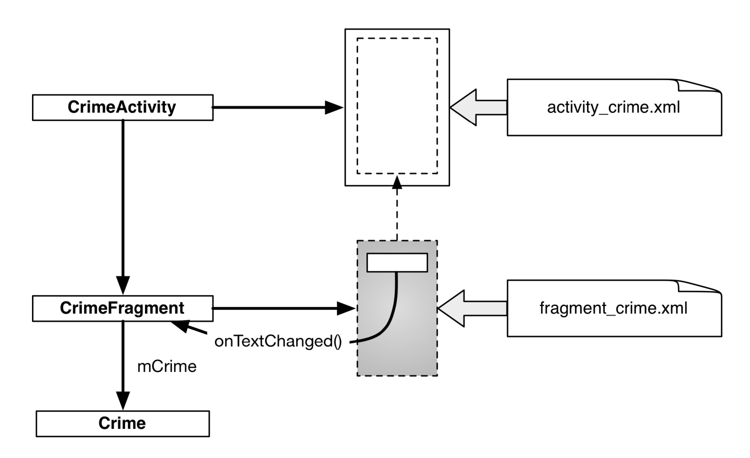


Рис. 7.5. CrimeActivity как хост CrimeFragment

Проект CriminalIntent будет большим; диаграмма объектов поможет понять логику его работы. На рис. 7.6 изображена общая структура CriminalIntent. Запоминать все объекты и связи между ними не обязательно, но прежде чем выходить в путь, полезно хотя бы в общих чертах понимать, куда вы направляетесь.

Мы видим, что класс `CrimeFragment` делает примерно то же, что в GeoQuiz делали активности: он создает пользовательский интерфейс и управляет с ним, а также обеспечивает взаимодействие с объектами модели.

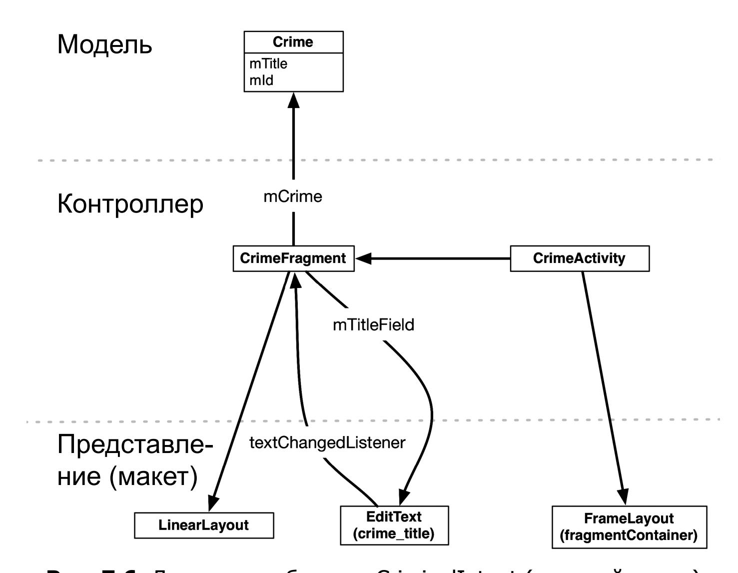


Рис. 7.6. Диаграмма объектов CriminalIntent (для этой главы)

Мы напишем три класса, изображенных на рис. 7.6: `Crime`, `CrimeFragment` и `CrimeActivity`.

Экземпляр `Crime` представляет одно офисное преступление. В этой главе описание преступления будет состоять только из заголовка и идентификатора. Заголовок содержит содержательный текст (например, «Свалка химических отходов в раковине» или «Кто-то украл мой йогурт!»), а идентификатор однозначно идентифицирует экземпляр `Crime`.

В этой главе мы для простоты будем использовать один экземпляр `Crime`. В класс `CrimeFragment` включается поле (`mCrime`) для хранения этого отдельного инцидента.

Представление `CrimeActivity` состоит из элемента `FrameLayout`, определяющего место, в котором будет отображаться представление `CrimeFragment`.

Представление `CrimeFragment` будет состоять из элементов `LinearLayout` и `EditText`. `CrimeFragment` определяет поле для виджета `EditText` (`mTitleField`) и назначает для него слушателя, обновляющего уровень модели при изменении текста.

Создание нового проекта

Но довольно разговоров; пора построить новое приложение. Создайте новое приложение Android (New ▸ Android Application Project). Введите имя приложения `CriminalIntent` и имя пакета `com.bignerdranch.android.criminalintent`, как показано на рис. 7.7. Укажите для построения новейшие версии API и убедитесь в том, что приложение совместимо с устройствами на базе Froyo.

Рис. 7.7. Создание приложения CriminalIntent

В следующем диалоговом окне снимите флажок, чтобы создать пользовательский значок лаунчера, и щелкните на кнопке Next. Выберите создание активности на базе пустого шаблона активности и щелкните на кнопке Next.

Наконец, введите имя активности `CrimeActivity` и щелкните на кнопке Finish (рис. 7.8).

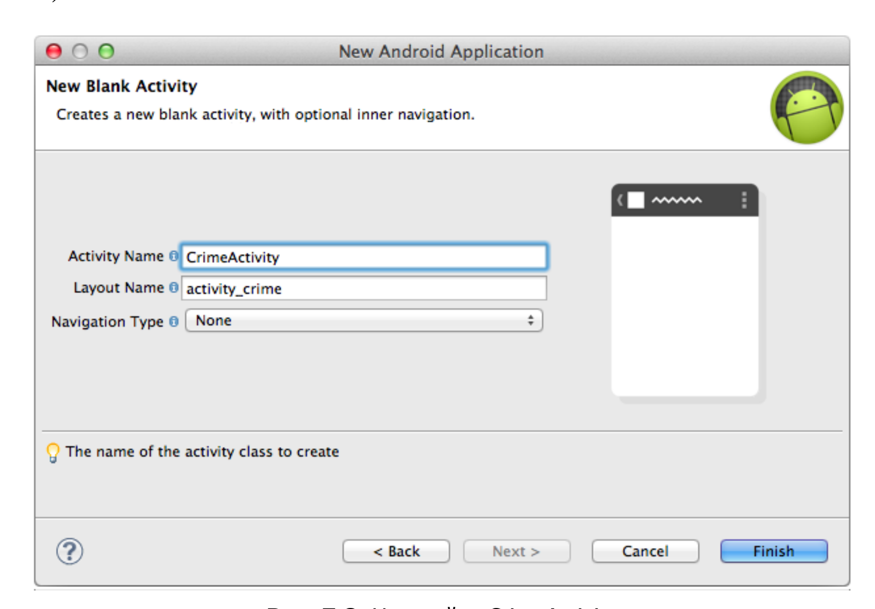


Рис. 7.8. Настройка CrimeActivity

Фрагменты и библиотека поддержки

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и неожиданной потребностью в гибкости пользовательского интерфейса. При создании `CriminalIntent` был указан минимальный обязательный SDK уровня 8, поэтому мы должны обеспечить совместимость приложения со старыми версиями Android.

К счастью, с фрагментами обратная совместимость достигается относительно легко: достаточно использовать классы фрагментов из библиотеки поддержки Android.

Библиотека поддержки (support library) уже является частью вашего проекта. Она находится в файле libs/android-support-v4.jar, куда была добавлена шаблоном. Библиотека поддержки включает класс `Fragment` (`android.support.v4.app.Fragment`), который может использоваться на любом устройстве Android с API уровня 4 и выше.

Класс из библиотеки поддержки используется не только в старых версиях при отсутствии «родного» класса; он также используется вместо него в новых версиях.

В библиотеку поддержки также входит важный класс `FragmentActivity` (`android.support.v4.app.FragmentActivity`). Для использования фрагментов нужны активности, которые умеют управлять фрагментами. В Honeycomb и последующих версиях

Android все субклассы Activity поддерживают управление фрагментами. В более ранних версиях Activity ничего не знает о фрагментах — как и ваши субклассы Activity. Для совместимости с более низкими уровнями API следует субклассировать FragmentActivity — субкласс Activity, предоставляющий функциональность управления фрагментами нового класса Activity даже в старых версиях Android.

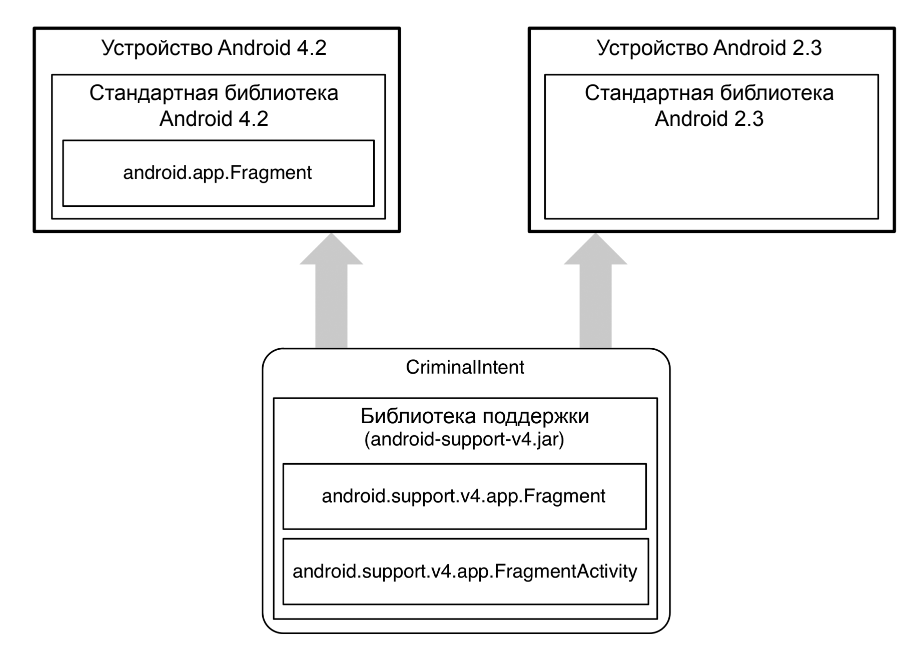


Рис. 7.9. Местонахождение разных классов фрагментов

На рис. 7.9 приведены имена этих классов и их местонахождение. Поскольку библиотека поддержки (и android.support.v4.app.Fragment) находится в вашем приложении, ее можно безопасно использовать независимо от того, где работает приложение.

На панели Package Explorer найдите и откройте файл CrimeActivity.java. Замените суперкласс CrimeActivity на FragmentActivity. Заодно удалите шаблонную реализацию onCreateOptionsMenu(Menu). (Мы создадим меню CriminalIntent «с нуля» в главе 16.)

Листинг 7.1. Настройка шаблонного кода (CrimeActivity.java)

```
public class CrimeActivity extends Activity FragmentActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

продолжение ⇗

Листинг 7.1 (*продолжение*)

```
        getMenuInflater().inflate(R.menu.activity_crime, menu);
        return true;
    }
}
```

Прежде чем продолжать с CrimeActivity, мы создадим уровень модели CriminalIntent. Для этого мы напишем класс Crime.

Создание класса Crime

На панели Package Explorer щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent и выберите команду New ▸ Class. Введите имя класса Crime, оставьте суперкласс java.lang.Object и щелкните на кнопке Finish.

Добавьте в Crime.java следующий код.

Листинг 7.2. Добавление кода в класс Crime (Crime.java)

```
public class Crime {

    private UUID mId;
    private String mTitle;

    public Crime() {
        // Генерирование уникального идентификатора
        mId = UUID.randomUUID();
    }
}
```

Затем для свойства mId, доступного только для чтения, необходимо сгенерировать только get-метод, а для свойства mTitle — get- и set-методы.

Щелкните правой кнопкой мыши после конструктора и выберите команду Source ▸ Generate Getters and Setters. Чтобы сгенерировать только get-метод для mId, щелкните на стрелке слева от имени переменной, чтобы раскрыть список методов, и установите флажок только рядом с getId(), как показано на рис. 7.10.

Листинг 7.3. Сгенерированные get- и set-методы (Crime.java)

```
public class Crime {
    private UUID mId;

    private String mTitle;

    public Crime() {
        mId = UUID.randomUUID();
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
```

```
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }
}
```

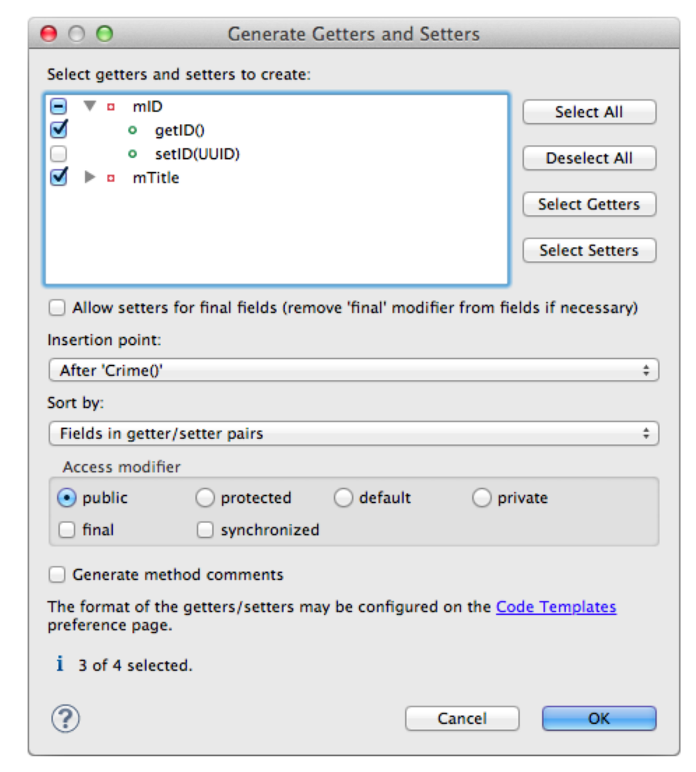


Рис. 7.10. Генерирование двух get-методов и одного set-метода

Вот и все, что нам понадобится для класса `Crime` и уровня модели CriminalIntent в этой главе.

Итак, мы создали уровень модели и активность, которая выполняет хостинг фрагмента в режиме совместимости с версиями Froyo и Gingerbread. А теперь более подробно рассмотрим, как активность выполняет свои функции хоста.

Хостинг UI-фрагментов

Чтобы стать хостом для UI-фрагмента, активность должна:

- определить место представления фрагмента в своем макете;
- управлять жизненным циклом экземпляра фрагмента.

Жизненный цикл фрагмента

На рис. 7.11 показан жизненный цикл фрагмента. Он имеет много общего с жизненным циклом активности: он тоже может находиться в состоянии остановки, приостановки и выполнения, он тоже содержит методы, переопределяемые для выполнения операций в критических точках, — многие из которых соответствуют методам жизненного цикла активности.

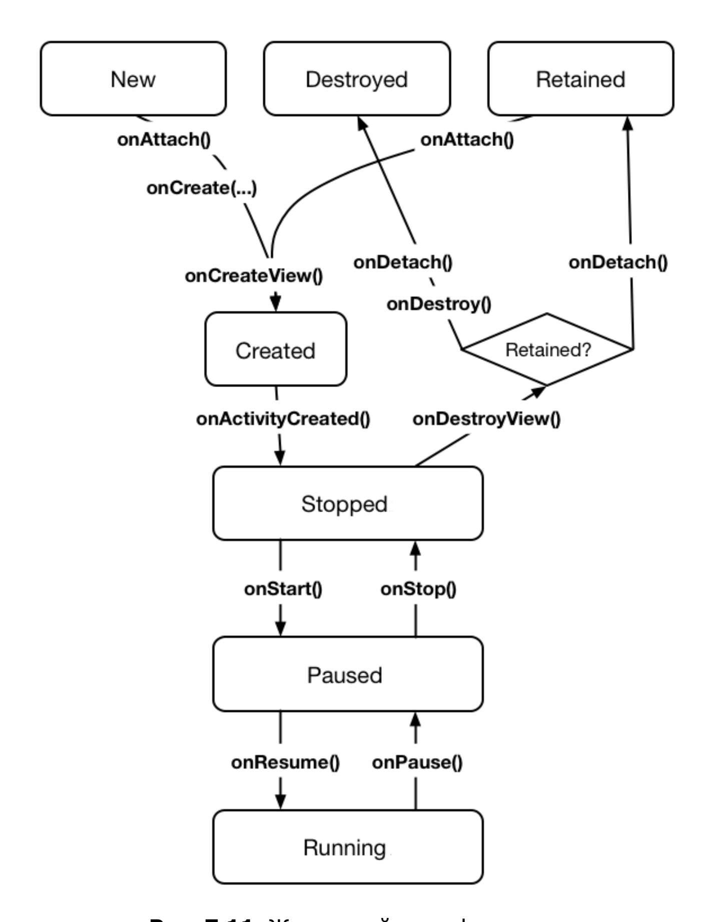


Рис. 7.11. Жизненный цикл фрагмента

Эти соответствия играют важную роль. Так как фрагмент работает по поручению активности, его состояние должно отражать текущее состояние активности. Следовательно, ему необходимые соответствующие методы жизненного цикла для выполнения работы активности.

Принципиальное различие между жизненными циклами фрагмента и активности заключается в том, что методы жизненного цикла фрагмента вызываются активностью-хостом, а не ОС. ОС ничего не знает о фрагментах, используемых активностью; фрагменты — внутреннее дело самой активности.

Методы жизненного цикла фрагментов будут более подробно рассмотрены, когда мы продолжим строить приложение CriminalIntent.

Два способа организации хостинга

Существует два основных способа организации хостинга UI-фрагментов в активности:

- добавление фрагмента в *макет* активности;
- добавление фрагмента в *код* активности.

Первый способ — с использованием так называемых *макетных фрагментов* — прост, но недостаточно гибок. Включение фрагмента в макет активности означает жесткую привязку фрагмента и его представления к представлению активности, и вам уже не удастся переключить этот фрагмент на протяжении жизненного цикла активности.

Несмотря на недостаток гибкости, макетные фрагменты полезны, и мы еще вернемся к ним в главе 13.

Второй способ сложнее, но только он позволяет управлять фрагментами во время выполнения. Разработчик сам определяет, когда фрагмент добавляется в активность и что с ним происходит после этого. Он может удалить фрагмент, заменить его другим фрагментом, а потом снова вернуть первый.

Итак, для достижения настоящей гибкости пользовательского интерфейса необходимо добавить фрагмент в коде. Именно этот способ мы используем для того, чтобы сделать `CrimeActivity` хостом `CrimeFragment`. Подробности реализации кода будут описаны позднее в этой главе, но сначала мы определим макет `CrimeActivity`.

Определение контейнерного представления

Мы добавим UI-фрагмент в коде активности-хоста, но мы все равно должны найти место для представления фрагмента в иерархии представлений активности. В макете `CrimeActivity` этим местом будет элемент `FrameLayout`, изображенный на рис. 7.12.

FrameLayout
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:id="**@+id/fragmentContainer**"
android:layout_width="**match_parent**"
android:layout_height="**match_parent**"

Рис. 7.12. Хостинг фрагмента для CrimeActivity

Элемент `FrameLayout` станет *контейнерным представлением* для `CrimeFragment`. Обратите внимание: контейнерное представление абсолютно универсально; оно не привязывается к классу `CrimeFragment`. Мы можем (и будем) использовать один макет для хостинга разных фрагментов.

Найдите макет CrimeActivity в файле res/layout/activity_crime.xml. Откройте файл и замените макет по умолчанию элементом FrameLayout, изображенным на рис. 7.12. Разметка XML должна совпадать с приведенной в листинге 7.4.

Листинг 7.4. Создание контейнера фрагмента (activity_crime.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragmentContainer"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  />
```

Хотя activity_crime.xml состоит исключительно из контейнерного представления одного фрагмента, макет активности может быть более сложным; он может определять несколько контейнерных представлений, а также собственные виджеты.

Просмотрите файл макета или запустите CriminalIntent, чтобы проверить свой код. Вы увидите только пустой элемент FrameLayout, потому что CrimeActivity еще не выполняет функции хоста фрагмента.

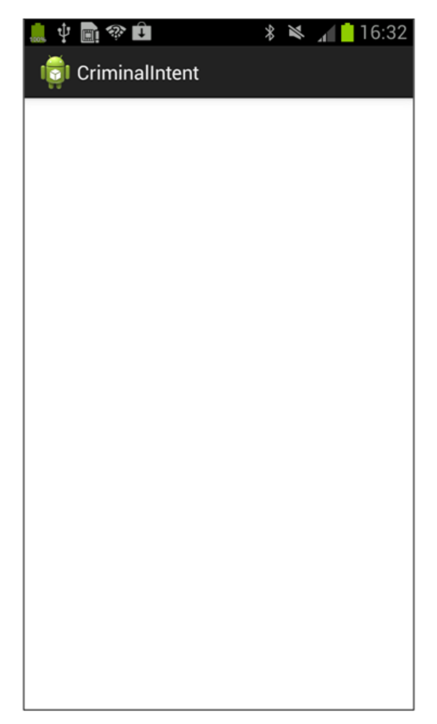


Рис. 7.13. Пустой элемент FrameLayout

Позднее мы напишем код, помещающий представление фрагмента в этот элемент FrameLayout. Но сначала фрагмент нужно создать.

Создание UI-фрагмента

Последовательность действий по созданию UI-фрагмента не отличается от последовательности действий по созданию активности:

- построение интерфейса посредством определения виджетов в файле макета;
- создание класса и назначение макета, который был определен ранее, его представлением;
- подключение виджетов, заполненных на основании макета в коде.

Определение макета CrimeFragment

Представление `CrimeFragment` будет отображать информацию, содержащуюся в экземпляре `Crime`. Со временем в классе `Crime` и представлении класса `CrimeFragment` добавится много интересного, а в этой главе мы ограничимся текстовым полем для хранения заголовка.

На рис. 7.14 изображен макет представления `CrimeFragment`. Он состоит из вертикального элемента `LinearLayout`, содержащего `EditText` — виджет с областью для ввода и редактирования текста.

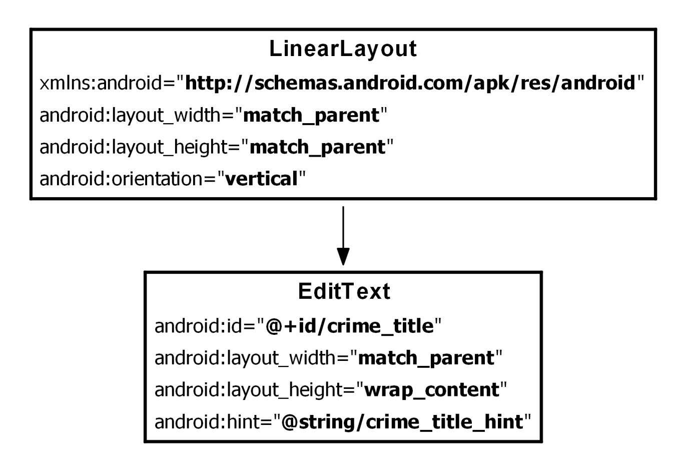


Рис. 7.14. Исходный макет CrimeFragment

Чтобы создать файл макета, щелкните правой кнопкой мыши на папке res/layout на панели Package Explorer и выберите команду New ▸ Android XML File. Проследите за тем, чтобы в окне был выбран тип ресурса Layout, а файлу фрагмента было присвоено имя fragment_crime.xml. Выберите корневым элементом `LinearLayout` и нажмите кнопку Finish.

Когда файл откроется, перейдите к разметке XML. Мастер уже добавил элемент `LinearLayout` за вас. Руководствуясь рис. 7.14, внесите необходимые изменения в fragment_crime.xml. Проверьте результаты по листингу 7.5.

Листинг 7.5. Файл макета для представления фрагмента (fragment_crime.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  >
  <EditText android:id="@+id/crime_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/crime_title_hint"
    />
</LinearLayout>
```

Откройте файл res/values/strings.xml, добавьте строковый ресурс `crime_title_hint` и уалите лишние строковые ресурсы `hello_world` и `menu_settings`, сгенерированные шаблоном.

Листинг 7.6. Добавление и удаление строк (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">CriminalIntent</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="title_activity_crime">CrimeActivity</string>
  <string name="crime_title_hint">Enter a title for the crime.</string>
</resources>
```

Сохраните файлы. Удаление строки `menu_settings` привело к возникновению ошибки в проекте. Чтобы исправить ее, найдите на панели Package Explorer файл res/menu/activity_crime.xml. Этот файл содержит определение меню, сгенерированное шаблоном, содержащее ссылку на строку `menu_settings`. Мы не будем использовать этот файл меню для CriminalIntent, поэтому его можно просто удалить с панели Package Explorer.

Удаление ресурса меню заставляет Eclipse построить приложение заново. Теперь проект должен быть свободен от ошибок. Переключитесь на графический конструктор, чтобы просмотреть результат разметки fragment_crime.xml.

Создание класса CrimeFragment

Щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent и выберите команду New ▸ Class. Введите имя класса `CrimeFragment` и щелкните на кнопке Browse, чтобы выбрать суперкласс. В открывшемся окне начинайте вводить строку Fragment. Мастер предлагает несколько классов. Выберите класс `android.support.v4.app.Fragment` — класс `Fragment` из библиотеки поддержки. Щелкните на кнопке OK.

(Если вы видите несколько версий `android.support.v4.app.Fragment`, выберите ту, которая является частью проекта CriminalIntent из CriminalIntent/libs/.)

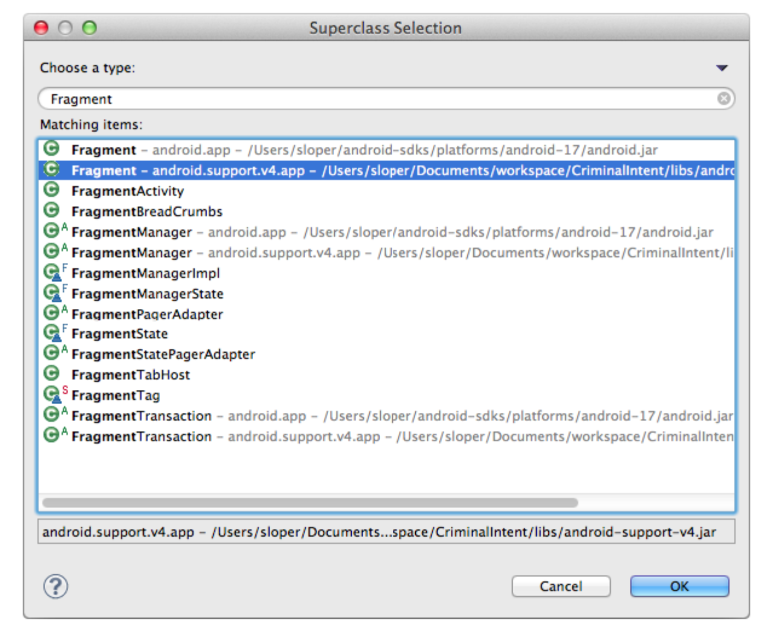


Рис. 7.15. Выбор класса Fragment из библиотеки поддержки

Реализация методов жизненного цикла фрагмента

`CrimeFragment` — контроллер, взаимодействующий с объектами модели и представления. Его задача — выдача подробной информации о конкретном преступлении и ее обновление при модификации пользователем.

В приложении GeoQuiz активности выполняли большую часть работы контроллера в методах жизненного цикла. В приложении CriminalIntent эта работа будет выполняться фрагментами в методах жизненного цикла фрагментов. Многие из этих методов соответствуют уже известным вам методам `Activity`, таким как `onCreate(Bundle)`.

В файле CrimeFragment.java добавьте переменную для экземпляра `Crime` и реализацию `Fragment.onCreate(Bundle)`.

Листинг 7.7. Переопределение Fragment.onCreate(Bundle) (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }
}
```

В этой реализации стоит обратить внимание на пару моментов. Во-первых, метод `Fragment.onCreate(Bundle)` объявлен открытым, тогда как метод `Activity.onCreate(Bundle)` объявлен защищенным. `Fragment.onCreate(…)` и другие методы жизненного цикла `Fragment` должны быть открытыми, потому что они будут вызываться произвольной активностью, которая станет хостом фрагмента.

Во-вторых, как и в случае с активностью, фрагмент использует объект `Bundle` для сохранения и загрузки состояния. Вы можете переопределить `Fragment.onSaveInstanceState(Bundle)` для ваших целей, как и метод `Activity.onSaveInstanceState(Bundle)`.

Обратите внимание на то, что *не происходит* в `Fragment.onCreate(…)`: мы не заполняем представление фрагмента. Экземпляр фрагмента настраивается в `Fragment.onCreate(…)`, но создание и настройка представления фрагмента осуществляются в другом методе жизненного цикла фрагмента:

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState)
```

Именно в этом методе заполняется макет представления фрагмента, а заполненный объект `View` возвращает активность хосту. Параметры `LayoutInflater` и `ViewGroup` необходимы для заполнения макета. Объект `Bundle` содержит данные, которые используются методом для воссоздания представления по сохраненному состоянию.

В файле CrimeFragment.java добавьте реализацию `onCreateView(…)`, которая заполняет разметку fragment_crime.xml.

Листинг 7.8. Переопределение onCreateView(...) (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        return v;
    }

}
```

В методе `onCreateView(…)` мы явно заполняем представление фрагмента, вызывая `LayoutInflater.inflate(…)` с передачей идентификатора ресурса макета. Второй параметр определяет родителя представления, что обычно необходимо для правильной настройки виджета. Третий параметр указывает, нужно ли включать заполненное представление в родителя. Мы передаем `false`, потому что представление будет добавлено в коде активности.

Подключение виджетов в фрагменте

В методе `onCreateView(…)` также настраивается реакция виджета `EditText` на ввод пользователя. После того как представление будет заполнено, метод получает ссылку на `EditText` и добавляет слушателя.

Листинг 7.9. Настройка виджета EditText (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            public void onTextChanged(
                    CharSequence c, int start, int before, int count) {
                mCrime.setTitle(c.toString());
            }

            public void beforeTextChanged(
                    CharSequence c, int start, int count, int after) {
                // Здесь намеренно оставлено пустое место
            }

            public void afterTextChanged(Editable c) {
                // И здесь тоже
            }
        });

        return v;
    }
}
```

Получение ссылок в `Fragment.onCreateView(…)` происходит практически так же, как в `Activity.onCreate(…)`. Единственное различие заключается в том, что для представления фрагмента вызывается метод `View.findViewById(int)`. Метод `Activity.findViewById(int)`, который мы использовали ранее, является вспомогательным методом, который вызывает `View.findViewById(int)` в своей внутренней реализации. У класса `Fragment` аналогичного вспомогательного метода нет, поэтому приходится вызывать основной метод.

Назначение слушателей в фрагменте работает точно так же, как в активности. В листинге 7.9 мы создаем анонимный класс, который реализует интерфейс слушателя `TextWatcher`. Этот интерфейс содержит три метода, но нас интересует только один: `onTextChanged(…)`.

В методе `onTextChanged(…)` мы вызываем `toString()` для объекта `CharSequence`, представляющего ввод пользователя. Этот метод возвращает строку, которая затем используется для задания заголовка `Crime`.

Код `CrimeFragment` готов. Было бы замечательно, если бы вы могли запустить CriminalIntent и поэкспериментировать с написанным кодом. К сожалению, это невозможно — фрагменты не могут выводить свои представления на экран.

Чтобы реализовать задуманное, необходимо сначала добавить `CrimeFragment` в `CrimeActivity`.

Добавление UI-фрагмента в FragmentManager

Когда в Honeycomb появился класс `Fragment`, в класс `Activity` были внесены изменения: в него был добавлен компонент, называемый `FragmentManager`. Он отвечает за управление фрагментами и добавление их представлений в иерархию представлений активности.

`FragmentManager` управляет двумя структурами: списком фрагментов и стеком транзакций фрагментов (о котором я вскоре расскажу).

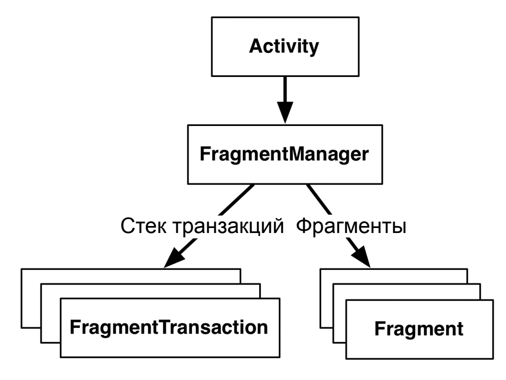


Рис. 7.16. FragmentManager

В приложении CriminalIntent нас интересует только список фрагментов `FragmentManager`.

Чтобы добавить фрагмент в активность в коде, следует обратиться с вызовом к объекту `FragmentManager` активности.

Прежде всего необходимо получить сам объект `FragmentManager`. В CrimeActivity.java включите следующий код в `onCreate(…)`.

Листинг 7.10. Получение объекта FragmentManager (CrimeActivity.java)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
    }
}
```

Мы вызываем getSupportFragmentManager(), потому что в приложении используется библиотека поддержки и класс FragmentActivity. Если бы нас не интересовала совместимость с устройствами, предшествующими Honeycomb, то вместо этого можно было бы субклассировать Activity и вызвать getFragmentManager().

Транзакции фрагментов

После получения объекта FragmentManager добавьте следующий код, который передает ему фрагмент для управления. (Позднее мы рассмотрим этот код более подробно, а пока просто включите его в приложение.)

Листинг 7.11. Добавление CrimeFragment (CrimeActivity.java)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

Разбираться в коде, добавленном в листинге 7.11, лучше всего не с начала. Найдите операцию add(…) и окружающий ее код. Этот код создает и закрепляет *транзакцию фрагмента*.

Листинг 7.12. Транзакция фрагмента (CrimeActivity.java)

```
        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
```

Транзакции фрагментов используются для добавления, удаления, присоединения, отсоединения и замены фрагментов в списке фрагментов. Они лежат в основе механизма использования фрагментов для формирования и модификации экранов во время выполнения. FragmentManager ведет стек транзакций, по которому вы можете перемещаться.

Метод FragmentManager.beginTransaction() создает и возвращает экземпляр FragmentTransaction. Класс FragmentTransaction использует динамичный интерфейс — методы, настраивающие FragmentTransaction, возвращают

FragmentTransaction вместо void, что позволяет объединять их вызовы в цепочку. Таким образом, выделенный код в листинге 7.12 означает: «Создать новую транзакцию фрагмента, включить в нее одну операцию add, а затем закрепить».

Метод add(…) является основным содержанием транзакции. Он получает два параметра: идентификатор контейнерного представления и недавно созданный объект CrimeFragment. Идентификатор контейнерного представления вам должен быть знаком: это идентификатор ресурса элемента FrameLayout, определенного в файле activity_crime.xml. Идентификатор контейнерного представления выполняет две функции:

- Он сообщает FragmentManager, где в представлении активности должно находиться представление фрагмента.
- Он обеспечивает однозначную идентификацию фрагмента в списке FragmentManager.

Когда вам потребуется получить экземпляр CrimeFragment от FragmentManager, запросите его по идентификатору контейнерного представления.

Листинг 7.13. Получение существующего фрагмента по идентификатору контейнерного представления (CrimeActivity.java)

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
```

Может показаться странным, что FragmentManager идентифицирует CrimeFragment по идентификатору ресурса FrameLayout. Однако идентификация UI-фрагмента по идентификатору ресурса его контейнерного представления встроена в механизм работы FragmentManager.

Теперь мы можем кратко описать код, добавленный в листинг 7.11, от начала до конца.

Сначала мы запрашиваем у FragmentManager фрагмент с идентификатором контейнерного представления R.id.fragmentContainer. Если этот фрагмент уже находится в списке, FragmentManager возвращает его.

Почему фрагмент может уже находиться в списке? Вызов CrimeActivity.onCreate(…) может быть выполнен в ответ на *воссоздание* объекта CrimeActivity после его уничтожения из-за поворота или освобождения памяти. При уничтожении активности ее экземпляр FragmentManager сохраняет список фрагментов. При воссоздании активности новый экземпляр FragmentManager загружает список и воссоздает хранящиеся в нем фрагменты, чтобы все работало, как прежде.

С другой стороны, если фрагменты с заданным идентификатором контейнерного представления отсутствуют, значение fragment равно null. В этом случае мы

создаем новый экземпляр CrimeFragment и новую транзакцию, которая добавляет фрагмент в список.

Теперь CrimeActivity является хостом для CrimeFragment. Чтобы убедиться в этом, запустите приложение CriminalIntent. На экране отображается представление, определенное в файле fragment_crime.xml (рис. 7.17).

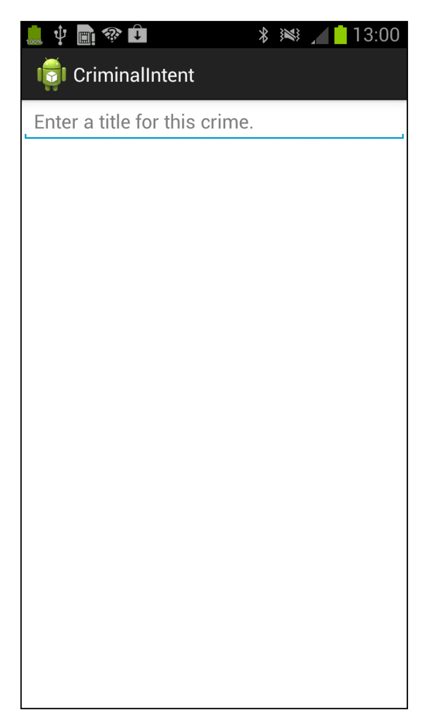


Рис. 7.17. CrimeActivity является хостом представления CrimeFragment

Возможно, один виджет на экране выглядит не таким уж большим достижением для всей работы, проделанной в этой главе. Однако мы заложили прочный фундамент для более серьезных задач, которые нам придется решать в приложении CriminalIntent в последующих главах.

FragmentManager и жизненный цикл фрагмента

После знакомства с FragmentManager стоит еще раз вернуться к жизненному циклу фрагмента.

Объект FragmentManager активности отвечает за вызов методов жизненного цикла фрагментов в списке. Методы onAttach(Activity), onCreate(Bundle) и onCreateView(…) вызываются при добавлении фрагмента в FragmentManager.

Метод onActivityCreated(…) вызывается после выполненния метода onCreate(…) активности-хоста. Мы добавляем CrimeFragment в CrimeActivity.onCreate(…), так что этот метод будет вызван после добавления фрагмента.

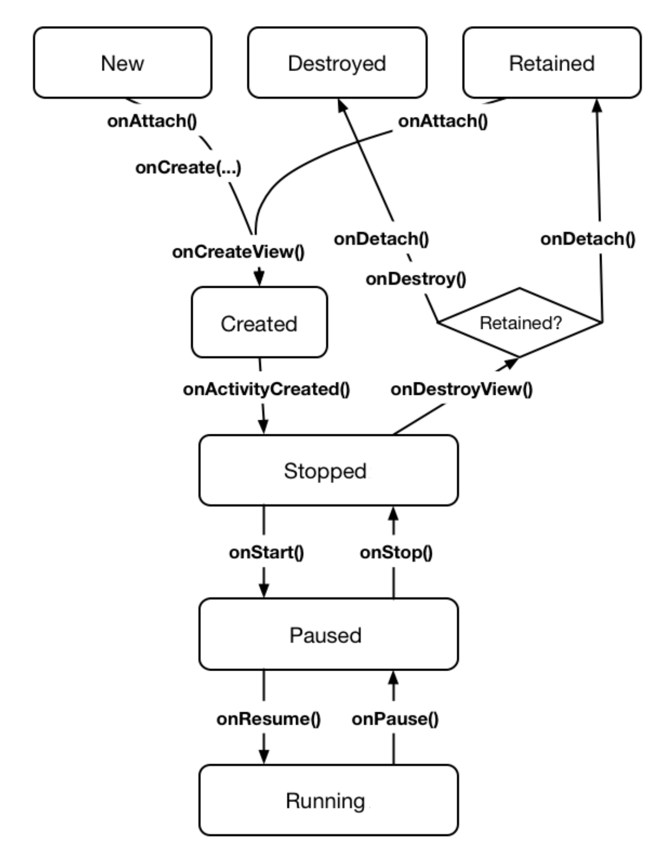


Рис. 7.18. Жизненный цикл фрагмента (повторно)

Что произойдет, если добавить фрагмент в то время, как активность уже находится в состоянии остановки, приостановки или выполнения? В этом случае `FragmentManager` немедленно проводит фрагмент через все действия, необходимые для его согласования с состоянием активности. Например, при добавлении фрагмента в активность, уже находящуюся в состоянии выполнения, фрагмент получит вызовы `onAttach(Activity)`, `onCreate(Bundle)`, `onCreateView(…)`, `onActivityCreated(Bundle)`, `onStart()` и затем `onResume()`.

После того как состояние фрагмента будет согласовано с состоянием активности, объект `FragmentManager` активности-хоста будет вызывать дальнейшие методы жизненного цикла приблизительно одновременно с получением соответствующих вызовов от ОС для синхронизации состояния фрагмента с состоянием активности. Нет никаких гарантий относительно того, будут ли методы фрагмента вызваны до или после методов активности.

При использовании библиотеки поддержки в жизненном цикле фрагмента появляется одно отличие: если добавить фрагмент в `Activity.onCreate(…)`, то метод `onActivityCreated(…)` не вызывается немедленно после `Activity.onCreate(…)`.

Вместо этого он вызывается при выполнении `Activity.onStart()`. Почему? В SDK до Honeycomb невозможно вызвать `onActivityCreated(…)` в правильный момент из `FragmentActivity`, поэтому метод вызывается при вызове следующего метода жизненного цикла. На практике это обычно ни на что не влияет; `onStart()` все равно вызывается немедленно после `Activity.onCreate(…)`.

Почему все наши активности используют фрагменты

С этого момента фрагменты будут использоваться во всех приложениях этой книги — даже самых простых. На первый взгляд такое решение кажется чрезмерным: многие примеры, которые вам встретятся в следующих главах, могут быть записаны без фрагментов. Для создания пользовательских интерфейсов и управления ими можно обойтись активностями; возможно, это даже уменьшит объем кода.

Тем не менее мы полагаем, что вам стоит поскорее привыкнуть к паттерну, который наверняка пригодится вам в реальной работе.

Кто-то скажет, что лучше сначала написать простое приложение без фрагментов, а потом добавить их, когда потребуется (и если потребуется). Эта идея заложена в основу методологии экстремального программирования YAGNI. Сокращение YAGNI означает «You Aren't Gonna Need It» («Вам это не понадобится»); этот принцип убеждает вас не писать код, который, по вашему мнению, может понадобиться позже. Почему? Потому что YAGNI. Возникает соблазн сказать YAGNI фрагментам.

К сожалению, добавление фрагментов в будущем может превратиться в мину замедленного действия. Превратить активность в активность хоста UI-фрагмента несложно, но при этом возникает множество неприятных ловушек. Если одними интерфейсами будут управлять активности, а другими — фрагменты, это только усугубит ситуацию, потому что вам придется отслеживать эти бессмысленные различия. Гораздо проще с самого начала написать код с использованием фрагментов и не возиться с его последующей переработкой, или же запоминать, какой стиль контроллера используется в каждой части вашего приложения.

Итак, в том, что касается фрагментов, мы применяем другой принцип: AUF, или «Always Use Fragments» («Всегда используйте фрагменты»). Выбирая между использованием фрагмента и активности, вы потратите немало нервных клеток, а это просто не стоит того. AUF!

Для любознательных: разработка для Honeycomb, ICS, Jelly Bean и т. д.

В этой главе вы узнали, как использовать библиотеку поддержки для включения фрагментов в проект с минимальной версией SDK ниже API уровня 11. Но если разработка предназначена исключительно для новых версий SDK, использовать

библиотеку поддержки не обязательно. Вместо этого можно использовать «родные» классы фрагментов в стандартной библиотеке.

Чтобы использовать классы фрагментов стандартной библиотеки, необходимо внести в проект четыре изменения:

- Задайте цель построения и минимальную версию SDK приложения на API уровня 11 и выше.
- Субклассируйте класс Activity стандартной библиотеки (android.app.Activity) вместо FragmentActivity. В API уровня 11 и выше активности содержат готовую поддержку фрагментов.
- Субклассируйте android.app.Fragment вместо android.support.v4.app.Fragment.
- Чтобы получить объект FragmentManager, используйте вызов getFragmentManager() вместо getSupportFragmentManager().

8 Макеты и виджеты

В этой главе мы поближе познакомимся с макетами и виджетами, а также включим хранение даты и статуса в CriminalIntent.

Обновление Crime

Откройте файл Crime.java и добавьте два новых поля. Поле `Date` представляет дату преступления, а поле `mSolved` — признак того, было ли преступление раскрыто.

Листинг 8.1. Добавление полей в класс Crime (Crime.java)

```
public class Crime {
    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;

    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }
   ...
}
```

Инициализация переменной `Date` конструктором `Date` по умолчанию присваивает `mDate` текущую дату.

Затем сгенерируйте `get`- и `set`-методы для своих новых полей (Source ▸ Generate Getters and Setters...).

Листинг 8.2. Сгенерированные get- и set-методы (Crime.java)

```
public class Crime {
   ...
    public void setTitle(String title) {
            mTitle = title;
      }
```

продолжение ⇨

Листинг 8.2 (*продолжение*)

```
    public Date getDate() {
        return mDate;
    }
    public void setDate(Date date) {
        mDate = date;
    }
    public boolean isSolved() {
        return mSolved;
    }
    public void setSolved(boolean solved) {
        mSolved = solved;
    }
}
```

Наши следующие действия — обновление макета в fragment_crime.xml новыми виджетами и их связывание с виджетами в CrimeFragment.java.

Обновление макета

Вот как будет выглядеть представление `CrimeFragment` к концу этой главы.

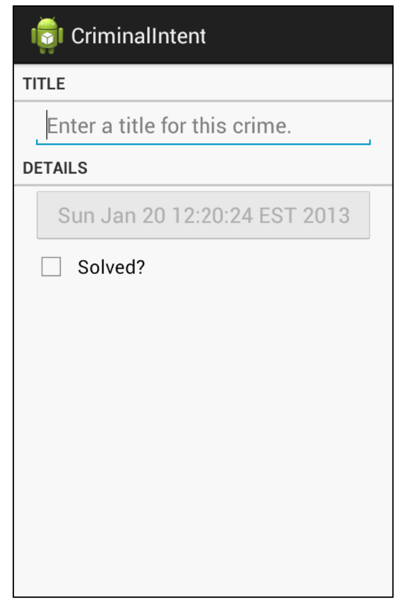


Рис. 8.1. CriminalIntent, эпизод 2

Чтобы добраться до этого экрана, мы добавим в макет `CrimeFragment` четыре виджета: два виджета `TextView`, `Button` и `CheckBox`.

Откройте файл fragment_crime.xml и внесите изменения, представленные в листинге 8.3. Возможно, вы получите ошибки отсутствия строковых ресурсов — вскоре мы их создадим.

Листинг 8.3. Добавление новых виджетов (fragment_crime.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  >
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_title_label"
    style="?android:listSeparatorTextViewStyle"
    />
  <EditText android:id="@+id/crime_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:hint="@string/crime_title_hint"
    />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_details_label"
    style="?android:listSeparatorTextViewStyle"
    />
  <Button android:id="@+id/crime_date"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    />
  <CheckBox android:id="@+id/crime_solved"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:text="@string/crime_solved_label"
    />
</LinearLayout>
```

Обратите внимание: для виджета `Button` мы не указали атрибут `android:text`. Кнопка будет выводить дату, хранящуюся в `Crime`, а ее текст будет задаваться в коде.

Почему дата выводится на `Button`? Это заготовка на будущее. Сейчас в качестве даты преступления по умолчанию используется текущая дата и ее нельзя изменить. В главе 12 мы изменим кнопку так, что при ее нажатии будет вызываться виджет `DatePicker`, при помощи которого пользователь сможет выбрать другую дату.

У макета имеются другие особенности, заслуживающие упоминания, например атрибут `style` и атрибуты `margin`. Но сначала мы заставим CriminalIntent работать с новыми виджетами.

Вернитесь к файлу res/values/strings.xml и добавьте необходимые строковые ресурсы.

Листинг 8.4. Добавление строковых ресурсов (strings.xml)

```
<resources>
  <string name="app_name">CriminalIntent</string>
  <string name="title_activity_crime">CrimeActivity</string>
  <string name="crime_title_hint">Enter a title for this crime.</string>
  <string name="crime_title_label">Title</string>
  <string name="crime_details_label">Details</string>
  <string name="crime_solved_label">Solved?</string>
</resources>
```

Сохраните файлы и проверьте возможные опечатки.

Подключение виджетов

Виджет `CheckBox` должен показывать, было ли преступление раскрыто. Изменение состояния `CheckBox` также должно приводить к обновлению поля `mSolved` класса `Crime`.

От `Button` пока что требуется только вывод даты из поля `mDate`.

В файле CrimeFragment.java добавьте две переменные экземпляра.

Листинг 8.5. Добавление переменных экземпляра для виджетов (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;

    @Override
    public void onCreate(Bundle savedInstanceState) {
    ...
```

Выполните организацию импорта, чтобы разрешить ссылки на `DatePicker` и `CheckBox`.

Затем в `onCreateView(…)` получите ссылку на новую кнопку, задайте в тексте кнопки дату преступления и заблокируйте ее.

Листинг 8.6. Назначение текста Button (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);
    ...

    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });

    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setEnabled(false);

    return v;
}
```

Блокировка кнопки гарантирует, что она не будет реагировать на нажатия. Также при этом изменяется оформление кнопки, чтобы сообщить пользователю о заблокированном состоянии. В главе 12 блокировка кнопки будет снята при назначении слушателя.

Теперь можно заняться CheckBox: мы получаем ссылку и назначаем слушателя, который будет обновлять поле mSolved объекта Crime.

Листинг 8.7. Назначение слушателя для изменений CheckBox (CrimeFragment.java)

```
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setEnabled(false);

    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
            // Назначение флага раскрытия преступления
            mCrime.setSolved(isChecked);
        }
    });

    return v;
}
```

При импортировании интерфейса OnCheckedChangeListener Eclipse предложит выбрать между интерфейсом, определенным в классе CompoundButton, и интерфейсом, определенным в классе RadioGroup. Выберите интерфейс CompoundButton; CheckBox является субклассом CompoundButton.

Если вы используете функцию автозавершения, над методом onCheckedChanged(…) может отображаться аннотация @Override, которой нет в листинге 8.7. На это отличие можно не обращать внимания. Для методов, определенных в интерфейсах, аннотации @Override не обязательны.

Запустите CriminalIntent. Переключите состояние CheckBox и насладитесь видом заблокированной кнопки, на которой отображается текущая дата.

Подробнее об атрибутах макетов XML

Вернемся к некоторым атрибутам, добавленным в файле fragment_crime.xml, и ответим на некоторые насущные вопросы по поводу виджетов и атрибутов.

Стили, темы и атрибуты тем

Стиль (style) представляет собой ресурс XML, который содержит атрибуты, описывающие внешний вид и поведение виджета. Например, ниже приведен ресурс стиля, который настраивает виджет на использование увеличенного размера текста.

```
<style name="BigTextStyle">
  <item name="android:textSize">20sp</item>
  <item name="android:layout_margin">3dp</item>
</style>
```

Вы можете создавать собственные стили (мы займемся этим в главе 24). Они добавляются в файл стилей из каталога res/values/, а ссылки на них в макетах выглядят так: `@style/my_own_style`.

Еще раз взгляните на виджеты `TextView` из файла fragment_crime.xml; каждый виджет имеет атрибут `style`, который ссылается на стиль, созданный Android. С этим конкретным стилем виджеты `TextView` выглядят как разделители списка, а берется он из *темы* приложения. *Тема* (theme) представляет собой набор стилей. Со структурной точки зрения тема сама является ресурсом стиля, атрибуты которого ссылаются на другие ресурсы стилей.

Android предоставляет платформенные темы, которые могут использоваться вашими приложениями. При создании CriminalIntent мастер предложил использовать тему приложения `Holo Light with Dark Action Bar`, и вы согласились.

Стиль из темы приложения можно применить к виджету при помощи *ссылки на атрибут темы* (theme attribute reference). Именно это мы делаем в файле fragment_crime.xml, используя значение `?android:listSeparatorTextViewStyle`.

Ссылка на атрибут темы приказывает менеджеру ресурсов Android: «Перейди к теме приложения и найди в ней атрибут с именем `listSeparatorTextViewStyle`. Этот атрибут указывает на другой ресурс стиля. Помести значение этого ресурса сюда».

Каждая тема Android включает атрибут с именем `listSeparatorTextViewStyle`, но его определение зависит от оформления конкретной темы. Использование ссылки на атрибут темы гарантирует, что оформление виджетов `TextView` будет соответствовать оформлению вашего приложения.

О том, как работают стили и темы, более подробно рассказано в главе 24.

Плотность пикселов, dp и sp

В файле fragment_crime.xml значение атрибута `margin` задается в единицах dp. Вы уже видели эти единицы в макетах; пришло время узнать, что они собой представляют.

Иногда значения атрибутов представления задаются в конкретных размерах (чаще всего в пикселах, но иногда в пунктах, миллиметрах или дюймах). Чаще всего этот способ используется для атрибутов размера текста, полей и отступов. Размер текста равен высоте текста в пикселах на экране устройства. Поля задают расстояния между представлениями, а отступы задают расстояние между внешней границей представления и его содержимым.

Android автоматически масштабирует изображения для разных плотностей пикселов экрана, используя содержимое каталогов drawable-ldpi, drawable-mdpi и drawable-hdpi. Но что произойдет, если ваши изображения масштабируются, а поля — нет? Или если пользователь выберет размер текста больше стандартного?

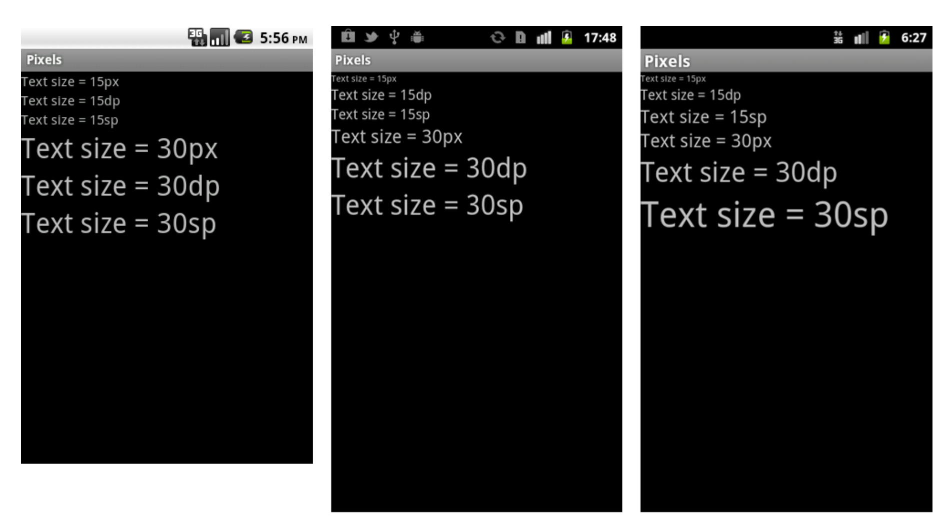


Рис. 8.2. Использование единиц устройства на TextView (*слева*: MDPI; *в середине:* HDPI; *справа*: HDPI с большим текстом)

Для решения таких проблем в Android поддерживаются единицы, не зависящие от плотности пикселов; используя их, можно получить одинаковые размеры на экранах с разными плотностями. Android преобразует эти единицы в пикселы во время выполнения, так что вам не придется самостоятельно заниматься сложными вычислениями:

- `dp` (или `dip`) — сокращение от «density-independent pixel» (пикселы, не зависящие от плотности); произносится «дип». Обычно эти единицы используются для полей, отступов и всего остального, для чего обычно задаются размеры в пикселах. На экранах с более высокой плотностью единицы dp разворачиваются в большее количество экранных пикселов. Одна единица dp всегда равна 1/160 дюйма на экране устройства. Размер будет одинаковым независимо от плотности пикселов.
- `sp` — сокращение от «scale-independent pixel» (пикселы, не зависящие от масштаба). Эти единицы, не зависящие от плотности пикселов устройства, также учитывают выбранный пользователем размер шрифта. Единицы sp почти всегда используются для назначения размера текста.
- `pt`, `mm`, `in` — масштабируемые единицы (как и `dp`), позволяющие задавать размеры интерфейсных элементов в пунктах (1/72 дюйма), миллиметрах или дюймах. Тем не менее мы не рекомендуем их использовать: не все устройства правильно настроены для правильного масштабирования этих устройств.

На практике и в этой книге почти исключительно используются только единицы dp и sp. Android преобразует эти значения в пикселы во время выполнения.

Рекомендации по проектированию интерфейсов Android

Для полей в нашем примере в листинге 8.3 используется значение `16dp`. Оно следует рекомендации по проектированию интерфейсов Android, известной как

«ритм 48dp». Полный список рекомендаций находится по адресу *http://developer.android.com/design/index.html*.

Современные приложения Android должны соответствовать этим рекомендациям, насколько это возможно. Рекомендации в значительной мере зависят от новой функциональности Android SDK, которая не всегда доступна или легко реализуема на старых устройствах. Некоторые рекомендации могут соблюдаться с использованием библиотеки поддержки, но для многих требуются библиотеки сторонних разработчиков — такие, как `ActionBarSherlock`, о которой можно прочитать в главе 16.

Параметры макета

Вероятно, вы уже заметили, что некоторые имена атрибутов начинаются с `layout_` (`android:layout_marginLeft`), а у других атрибутов этого префикса нет (`android:text`).

Атрибуты, имена которых не начинаются с `layout_`, являются рекомендациями для виджетов. При заполнении виджет вызывает метод для настройки своей конфигурации на основании этих атрибутов и их значений.

Если имя атрибута начинается с `layout_`, то этот атрибут является директивой для родителя этого виджета. Такие атрибуты, называемые *параметрами макета*, сообщают родительскому макету, как следует расположить дочерний элемент внутри родителя.

Даже если объект макета (например, `LinearLayout`) является корневым элементом, он все равно остается виджетом, имеющим родителя, и у него есть параметры макета. Определяя элемент `LinearLayout` в файле fragment_crime.xml, мы задали атрибуты `android:layout_width` и `android:layout_height`. Эти атрибуты будут использоваться родительским макетом `LinearLayout` при заполнении. В данном случае параметры макета `LinearLayout` будут использоваться элементом `FrameLayout` в представлении содержимого `CrimeActivity`.

Поля и отступы

В файле fragment_crime.xml виджетам назначаются атрибуты `margin` и `padding`. Начинающие разработчики иногда путают эти атрибуты, определяющие соответственно *поля* и *отступы*. Теперь, когда вы понимаете, что такое параметр макета, будет проще объяснить, чем они различаются. Атрибуты `margin` являются параметрами макета; они определяют расстояние между виджетами. Так как виджет располагает информацией только о самом себе, за соблюдение полей отвечает родитель виджета.

Напротив, отступ не является параметром макета. Атрибут `android:padding` сообщает виджету, с каким превышением размера содержимого он должен прорисовывать себя. Допустим, вы хотите заметно увеличить размеры кнопки даты без изменения размера текста. Добавьте в `Button` следующий атрибут, сохраните макет и запустите приложение заново.

Листинг 8.8. Назначение отступов

```
<Button android:id="@+id/crime_date"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginLeft="16dp"
  android:layout_marginRight="16dp"
  android:padding="80dp"
  />
```

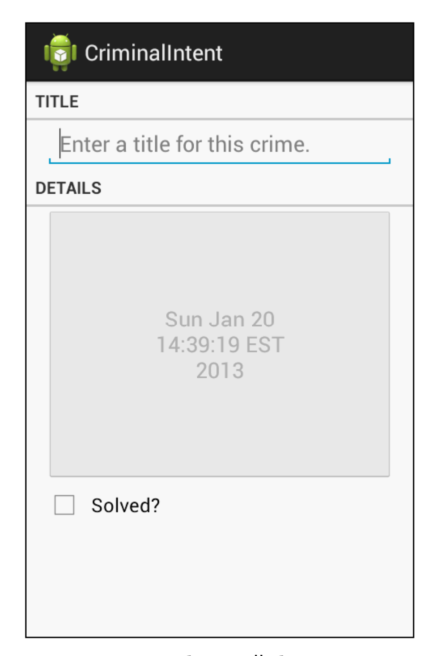


Рис. 8.3. Для любителей больших кнопок

Не забудьте удалить этот атрибут, прежде чем продолжать работу.

Использование графического конструктора

До настоящего момента мы создавали макеты, вводя разметку XML. В этом разделе мы используем графический конструктор для построения альтернативного альбомного макета CrimeFragment.

Большинство встроенных классов макетов — таких, как LinearLayout — автоматически изменяют размеры себя и своих потомков при поворотах. Однако в некоторых случаях изменение размеров по умолчанию недостаточно эффективно использует свободное пространство.

Запустите приложение CriminalIntent и поверните устройство, чтобы увидеть макет CrimeFragment в альбомной ориентации.

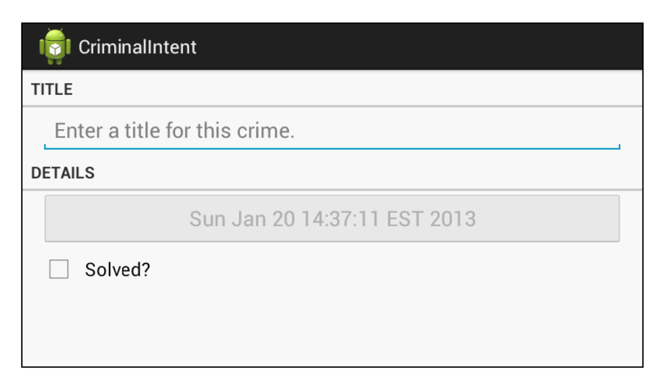


Рис. 8.4. CrimeFragment в альбомной ориентации

Кнопка даты становится слишком длинной; было бы лучше, если бы в альбомной ориентации кнопка и флажок располагались рядом друг с другом. Для этого создайте каталог res/layout-land (щелкните правой кнопкой мыши на каталоге res/ на панели Package Explorer и выберите команду New ▸ Folder), после чего скопируйте файл fragment_crime.xml в res/layout-land/.

Чтобы внести изменения в графическом конструкторе, сначала закройте файл res/layout/fragment_crime.xml, если он открыт в редакторе. Откройте файл res/layout-land/fragment_crime.xml и выберите вкладку Graphical Layout.

В середине графического конструктора макета в области предварительного просмотра отображается уже знакомый альбомный макет. Слева находится *палитра*. Панель содержит все необходимые виджеты, упорядоченные по категориям.

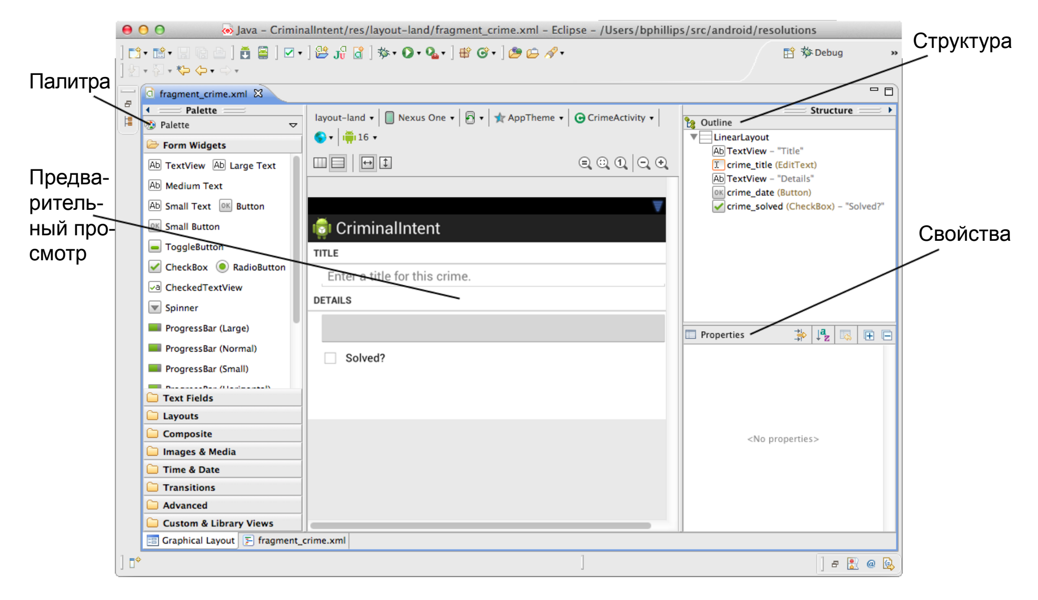


Рис. 8.5. Графический конструктор макета

Справа от области предварительного просмотра находится *панель структуры*. На ней показана иерархия виджетов в макете.

Под панелью структуры находится панель *свойств*. На ней можно просматривать и редактировать атрибуты виджета, выделенного на панели структуры.

Какие же изменения следует внести в этот макет? Взгляните на рис. 8.6.

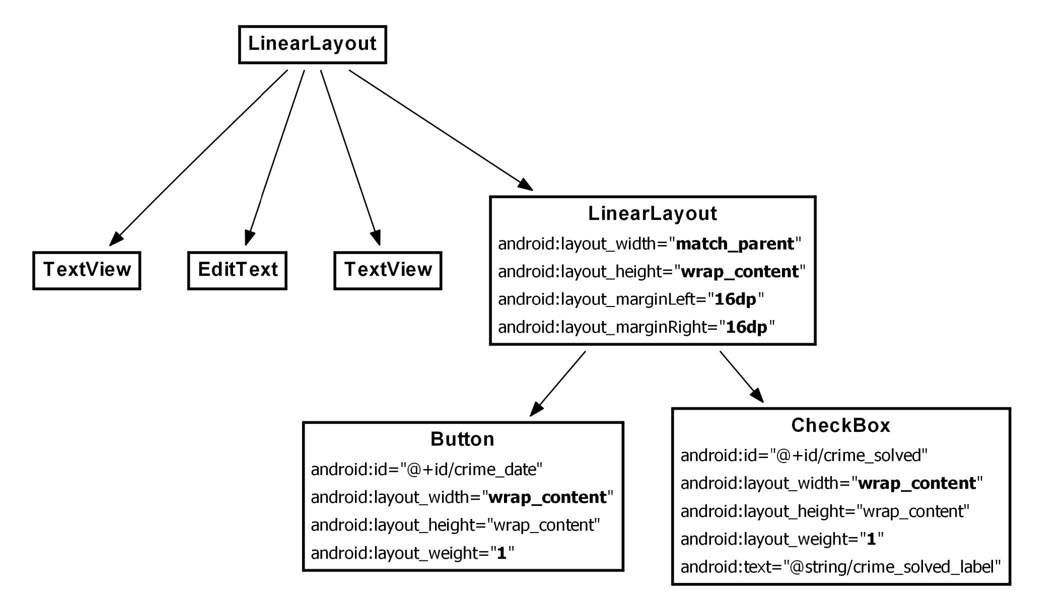


Рис. 8.6. Альбомный макет для CrimeFragment

Изменения можно разделить на четыре группы:

- Добавление виджета `LinearLayout` в макет.
- Редактирование атрибутов `LinearLayout`.
- Назначение виджетов `Button` и `CheckBox` потомками `LinearLayout`.
- Обновление параметров макета `Button` и `CheckBox`.

Добавление нового виджета

Чтобы добавить виджет, можно выделить его в палитре и перетащить на панель структуры. Щелкните на категории Layouts в палитре. Выберите пункт LinearLayout (Horizontal) и перетащите на панель структуры. Отпустите перетаскиваемый объект `LinearLayout` на коренном элементе `LinearLayout`, чтобы сделать его прямым потомком корневого элемента `LinearLayout`.

Виджеты также можно добавлять перетаскиванием из палитры в область предварительного просмотра. Однако виджеты `Layout` часто пусты или закрыты другими представлениями, поэтому может быть трудно понять, где следует разместить виджет в области предварительного просмотра для получения нужной иерархии.

Перетаскивание на панель структуры существенно упрощает выполнение этой операции.

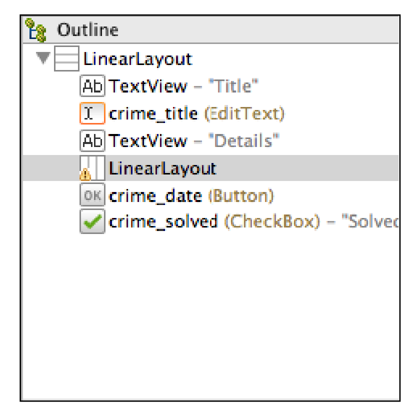


Рис. 8.7. Добавление виджета LinearLayout в файл fragment_crime.xml

Редактирование атрибутов в свойствах

Выберите новый виджет `LinearLayout` на панели структуры, чтобы отобразить его атрибуты на панели свойств. Разверните категорию Layout Parameters, затем категорию Margins.

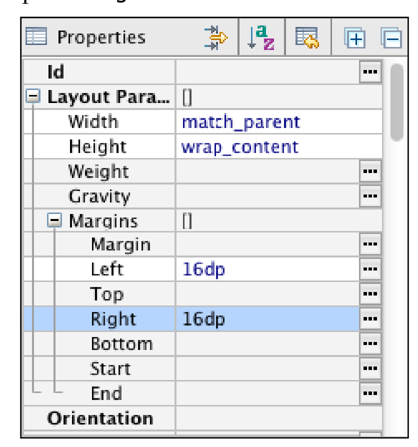


Рис. 8.8. Назначение полей на панели свойств

Мы хотим, чтобы поля нового виджета `LinearLayout` совпадали с полями других виджетов. Выделите поле рядом с Left и введите значение 16dp. Сделайте то же самое для правого (Right) поля.

(На некоторых платформах при попытке ввести данные в эти поля на экране появляется временное окно. Это обходное решение известной ошибки... которое, к сожалению, не позволяет ввести значение. Если это произойдет, сохраните файл, переключитесь на разметку XML и скопируйте два атрибута `margin` из `EditText` в `LinearLayout`).

Сохраните файл макета и переключитесь на разметку XML при помощи вкладки fragment_crime.xml в нижней части области предварительного просмотра. Вы увидите только что добавленный элемент `LinearLayout` с атрибутами полей.

Реорганизация виджетов на панели структуры

Следующий шаг — назначение виджетов Button и CheckBox потомками нового виджета LinearLayout. Вернитесь к графическому конструктору, выберите на панели структуры виджет Button и перетащите его на LinearLayout.

На панели структуры отражается тот факт, что Button теперь является потомком нового виджета LinearLayout. Проделайте то же самое cCheckBox.

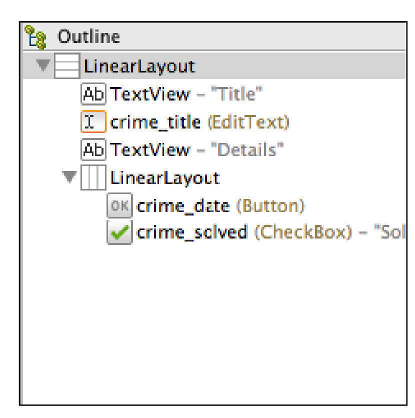


Рис. 8.9. Виджеты Button и CheckBox теперь являются потомками LinearLayout

Если потомки размещаются не в том порядке, их можно переупорядочить посредством перетаскивания. Также на панели структуры можно удалять виджеты из макета, но будьте осторожны: удаление виджета приводит к удалению его потомков.

В области предварительного просмотра виджет CheckBox не виден — его скрывает Button. Виджет LinearLayout проверил ширину (match_parent) своего первого потомка (Button) и выделил ему все пространство, ничего не оставив CheckBox (рис. 8.10).

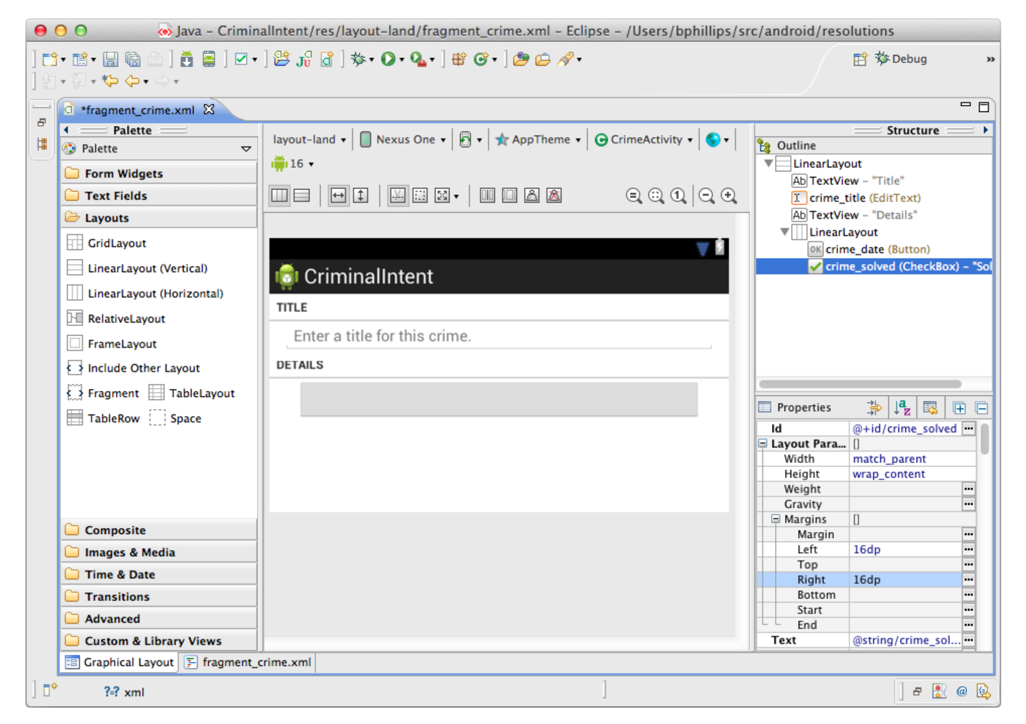


Рис. 8.10. Виджет Button, который был определен первым, скрывает CheckBox

Чтобы восстановить равноправие потомков LinearLayout, мы изменим параметры макета потомков.

Обновление параметров макета потомков

Сначала выделите кнопку даты на панели структуры. На панели свойств щелкните на текущем значении Width и замените его значением wrap_content.

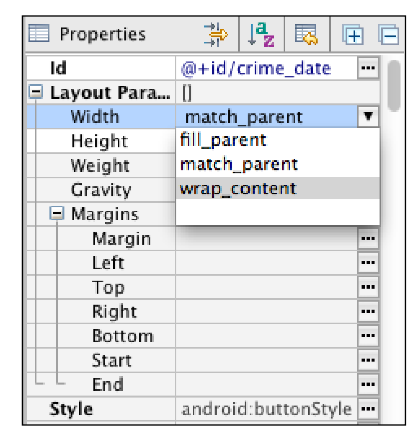


Рис. 8.11. Ширине кнопки задается значение wrap_content

Удалите оба значения 16dp полей кнопки. Теперь, когда кнопка находится внутри LinearLayout, поля ей не нужны.

Найдите поле Weight в разделе Layout Parameters и задайте ему значение 1. Это поле соответствует атрибуту android:layout_weight на рис. 8.6.

Выберите виджет CheckBox на панели структуры и внесите те же изменения: атрибут Width должен содержать wrap_content, атрибут Weight — 1, а атрибуты полей должны быть пустыми.

В области предварительного просмотра убедитесь в том, что оба виджета теперь видны. Сохраните файл и вернитесь к XML, чтобы подтвердить изменения. В листинге 8.9 приведена соответствующая разметка XML.

Листинг 8.9. Разметка XML макета, созданного в графическом конструкторе (layout-land/fragment_crime.xml)

```
  ...
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_details_label"
    style="?android:listSeparatorTextViewStyle"
    />
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp" >
      <Button
        android:id="@+id/crime_date"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
      <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/crime_solved_label" />
  </LinearLayout>
</LinearLayout>
```

Как работает android:layout_weight

Атрибут веса android:layout_weight сообщает виджету LinearLayout, как он должен распределить потомков по размеру контейнера. Обоим виджетам заданы одинаковые значения ширины, но это не гарантирует, что они будут иметь одинаковую ширину на экране. Для определения ширин дочерних представлений LinearLayout использует комбинацию параметров layout_width и layout_weight.

LinearLayout вычисляет ширину представления в два прохода. На первом проходе LinearLayout проверяет значение layout_width (или layout_height для вертикальной ориентации). Значение layout_width как для Button, так и для CheckBox теперь равно wrap_content, так что каждому представлению выделяется место, достаточное только для его прорисовки (рис. 8.12).

(По области предварительного просмотра трудно понять, как работает система весов, потому что содержимое кнопки в макет не входит. На следующих рисунках показано, как будет выглядеть LinearLayout, если кнопка уже имеет содержимое.)

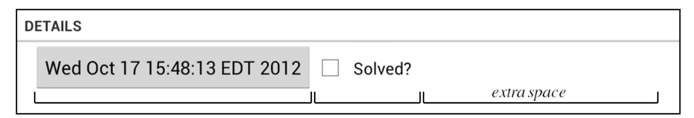


Рис. 8.12. Проход 1: распределение пространства на основании layout_width

На следующем проходе LinearLayout распределяет дополнительное пространство на основании значений layout_weight (рис. 8.13).

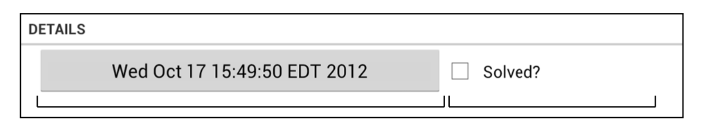


Рис. 8.13. Проход 2: распределение дополнительного пространства на основании layout_weight

В нашем макете Button и CheckBox имеют одинаковые значения layout_weight, поэтому дополнительное пространство распределяется в соотношении 50/50. Если задать весовой коэффициент Button равным 2, то ей будет выделено 2/3 дополнительного пространства, а CheckBox достанется всего 1/3 (рис. 8.14).

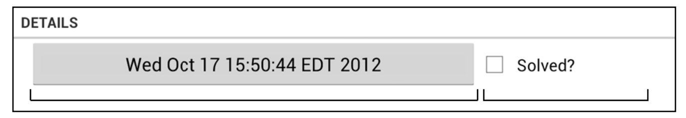


Рис. 8.14. Неравномерное распределение дополнительного пространства с пропорцией layout_weight 2:1

В качестве весового коэффициента может использоваться любое вещественное число. Программисты используют разные системы обозначений весов. В файле

fragment_crime.xml используется система «рецепт коктейля». Также часто применяются наборы весов, сумма которых составляет 1.0 или 100; в этом случае вес кнопки в приведенном примере составит 0.66 или 66 соответственно.

Что если вы хотите, чтобы виджет `LinearLayout` выделял для каждого представления ровно 50 % своей ширины? Просто пропустите первый проход, задав атрибуту `layout_width` каждого виджета значение `0dp` вместо `wrap_content`. В этом случае `LinearLayout` принимает решения только на основании значений `layout_weight` (рис. 8.15).

Рис. 8.15. При layout_width="0dp" учитываются только значения layout_weight

Графический конструктор макетов

Графический конструктор макетов удобен, и Android совершенствует его с каждым выпуском ADT. Однако порой он работает медленно и ненадежно и тогда проще использовать прямой ввод разметки XML. Вы можете переключаться между внесением изменений в графическом конструкторе и в XML (для надежности не забудьте сохранить файл перед переключением).

Не стесняйтесь использовать графический конструктор для создания макетов из этой книги. В дальнейшем, когда потребуется создать макет, мы будем приводить диаграмму наподобие рис. 8.6. Вы сами сможете решить, как создать ее — в виде разметки XML, в графическом конструкторе или сочетанием этих двух способов.

Идентификаторы виджетов и множественные макеты

Два макета, созданные для CriminalIntent, отличаются незначительно, но в некоторых ситуациях возможны более серьезные расхождения. В таких случаях перед обращением к виджету в коде необходимо убедиться в том, что он действительно существует.

Если виджет присутствует в одном макете и отсутствует в другом, то для проверки его наличия в текущей ориентации перед вызовом методов следует использовать проверку `null`:

```
    Button landscapeOnlyButton = (Button)v.findViewById(R.id.landscapeOnlyButton);
    if (landscapeOnlyButton != null) {
        // Операции
}
```

Наконец, помните, что для того чтобы ваш код мог найти виджет, последний должен иметь *одинаковые* значения атрибута `android:id` во всех макетах, в которых он присутствует.

Упражнение. Форматирование даты

Объект `Date` больше напоминает временную метку (timestamp), чем традиционную дату. При вызове `toString()` для `Date` вы получаете именно временную метку, которая отображается на кнопке. Временные метки хорошо подходят для отчетов, но на кнопке было бы лучше выводить дату в формате, более привычном для людей (например, «Oct 12, 2012»). Для этого можно воспользоваться экземпляром класса `android.text.format.DateFormat`. Хорошей отправной точкой в работе станет описание этого класса в документации Android.

Используйте методы класса `DateFormat` для формирования строки в стандартном формате или же подготовьте собственную форматную строку. Чтобы задача стала более творческой, попробуйте создать форматную строку для вывода дня недели («Tuesday, Oct 12, 2012»).

9 Вывод списков и ListFragment

Уровень модели CriminalIntent в настоящее время состоит из единственного экземпляра `Crime`. В этой главе мы обновим приложение CriminalIntent, чтобы оно поддерживало списки. В списке для каждого преступления будет отображаться краткое описание и дата, а также признак его раскрытия.

Рис. 9.1. Список преступлений

На рис. 9.2 показана общая структура приложения CriminalIntent для этой главы.

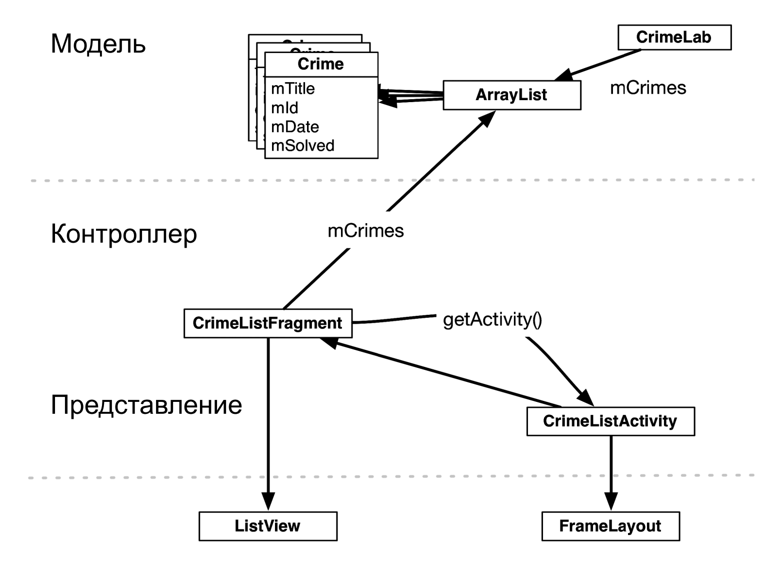


Рис. 9.2. Приложение CriminalIntent со списком

На уровне модели появляется новый объект `CrimeLab`, который представляет собой централизованное хранилище для объектов `Crime`.

Для отображения списка на уровне контроллера CriminalIntent появляется новая активность и новый фрагмент: `CrimeListActivity` и `CrimeListFragment`.

`CrimeListFragment` субклассирует `ListFragment` — субкласс `Fragment`, который обладает встроенными средствами для поддержки списков. Эти контроллеры взаимодействуют друг с другом и `CrimeLab` для обращения к данным уровня модели.

(Где находятся классы `CrimeActivity` и `CrimeFragment` на рис. 9.2? Они являются частью представления детализации, поэтому на рисунке их нет. В главе 10 мы свяжем части списка и детализации CriminalIntent.)

На рис. 9.2 также видны объекты представлений, связанные с `CrimeListActivity` и `CrimeListFragment`. Представление активности состоит из объекта `FrameLayout`, содержащего фрагмент. Представление фрагмента состоит из `ListView`. Взаимодействие между `ListFragment` и `ListView` более подробно рассматривается позднее в этой главе.

Обновление уровня модели CriminalIntent

Прежде всего необходимо преобразовать уровень модели CriminalIntent из одного объекта `Crime` в массив объектов `Crime`.

`ArrayList<E>` — класс Java, реализующий упорядоченный список объектов заданного типа. В нем содержатся методы извлечения, добавления и удаления элементов.

Синглеты и централизованное хранение данных

Для хранения массива-списка преступлений будет использоваться *синглетный* (singleton) класс. Такие классы допускают создание только одного экземпляра.

Экземпляр синглетного класса существует до тех пор, пока приложение остается в памяти, так что при хранении списка в синглетном объекте данные остаются доступными, что бы ни происходило с активностями, фрагментами и их жизненными циклами.

Чтобы создать синглетный класс, следует создать класс с закрытым конструктором и методом `get()`, который возвращает экземпляр. Если экземпляр уже существует, то `get()` просто возвращает его. Если экземпляр еще не существует, то `get()` вызывает конструктор для его создания.

Щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent и выберите команду New ▸ Class. Введите имя класса `CrimeLab` и щелкните на кнопке Finish.

В файле CrimeLab.java реализуйте `CrimeLab` как синглетный класс с закрытым конструктором и методом `get(Context)`.

Листинг 9.1. Синглетный класс (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;
    private Context mAppContext;

    private CrimeLab(Context appContext) {
        mAppContext = appContext;
    }

    public static CrimeLab get(Context c) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(c.getApplicationContext());
        }
        return sCrimeLab;
    }
}
```

Обратите внимание на префикс `s` у переменной `sCrimeLab`. Мы используем это условное обозначение Android, чтобы показать, что переменная `sCrimeLab` является статической.

Конструктору `CrimeLab` передается параметр `Context`. В Android такая ситуация встречается очень часто; наличие параметра `Context` позволяет синглетному классу запускать активности, обращаться к ресурсам проекта, находить закрытое хранилище вашего приложения и т. д.

Обратите внимание: в `get(Context)` параметр `Context` не передается конструктору. В `Context` может содержаться `Activity` или другой объект `Context` — например, `Service`. Мы не можем быть уверены в том, что произвольный объект `Context` будет существовать все то время, когда он может понадобиться `CrimeLab` — то есть на протяжении всего жизненного цикла приложения.

Чтобы гарантировать, что синглетному классу для работы будет доступен объект `Context` с долгим сроком жизни, мы вызываем `getApplicationContext()` и подменяем

переданный объект Context *контекстом приложения*. Контекст приложения глобален по отношению к вашему приложению. Если в вашем приложении задействован синглетный класс уровня приложения, всегда используйте контекст приложения.

Для начала предоставим CrimeLab несколько объектов Crime для хранения. В конструкторе CrimeLab создайте пустой массив ArrayList объектов Crime. Также добавьте два метода: getCrimes() возвращает список, а getCrime(UUID) возвращает объект Crime с заданным идентификатором (листинг 9.2).

Листинг 9.2. Создание списка ArrayList объектов Crime (CrimeLab.java)

```
public class CrimeLab {
    private ArrayList<Crime> mCrimes;

    private static CrimeLab sCrimeLab;
    private Context mAppContext;

    private CrimeLab(Context appContext) {
        mAppContext = appContext;
        mCrimes = new ArrayList<Crime>();
    }

    public static CrimeLab get(Context c) {
        ...
    }

    public ArrayList<Crime> getCrimes() {
        return mCrimes;
    }

    public Crime getCrime(UUID id) {
        for (Crime c : mCrimes) {
            if (c.getId().equals(id))
                return c;
        }
        return null;
    }
}
```

Со временем ArrayList будет содержать объекты Crime, созданные пользователем, которые будут сохраняться и загружаться повторно. А пока заполним массив 100 однообразными объектами Crime (листинг 9.3).

Листинг 9.3. Генерирование тестовых объектов (CrimeLab.java)

```
private CrimeLab(Context appContext) {
    mAppContext = appContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setSolved(i % 2 == 0); // Для каждого второго объекта
        mCrimes.add(c);
    }
}
```

Теперь у нас имеется полностью загруженный уровень модели и 100 преступлений для вывода на экран.

Создание ListFragment

Создайте новый класс с именем CrimeListFragment. Щелкните на кнопке Browse, чтобы выбрать суперкласс. Найдите и выберите строку ListFragment – android.support.v4.app, после чего щелкните на кнопке Finish, чтобы сгенерировать класс CrimeListFragment.

Класс ListFragment появился в Honeycomb, но он дублируется в библиотеке поддержки. Таким образом, с совместимостью проблем не будет — при условии, что вы используете класс библиотеки поддержки android.support.v4.app.ListFragment.

В файле CrimeListFragment.java переопределите метод onCreate(Bundle), чтобы задать заголовок активности, которая станет хостом данного фрагмента.

Листинг 9.4. Добавление метода onCreate(Bundle) в новую активность (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getActivity().setTitle(R.string.crimes_title);
    }

}
```

Обратите внимание на метод getActivity(). Этот вспомогательный метод Fragment возвращает активность-хоста и позволяет фрагменту более активно выполнять функции активности. В приведенном примере он используется для вызова Activity.setTitle(int), заменяющего содержимое панели действий (строки заголовка на старых устройствах) значением переданного строкового ресурса.

Мы не будем переопределять onCreateView(…) или заполнять макет CrimeListFragment. Реализация ListFragment по умолчанию заполняет макет, определяющий полноэкранный виджет ListView; пока мы будем использовать этот макет. В следующих главах мы переопределим CrimeListFragment.onCreateView(…) для расширения функциональности приложения.

Добавьте в файл strings.xml строковый ресурс для заголовка активности списка.

Листинг 9.5. Добавление строкового ресурса для заголовка новой активности (strings.xml)

```
  ...
  <string name="crime_solved_label">Solved?</string>
  <string name="crimes_title">Crimes</string>
</resources
```

Классу CrimeListFragment необходим доступ к списку преступлений, хранящемуся в CrimeLab. Включите в метод CrimeListFragment.onCreate(…) синглетный экземпляр CrimeLab и получите список преступлений.

Листинг 9.6. Обращение к данным списка в CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getActivity().setTitle(R.string.crimes_title);
        mCrimes = CrimeLab.get(getActivity()).getCrimes();
    }
}
```

Абстрактная активность для хостинга фрагмента

Вскоре мы создадим класс CrimeListActivity, предназначенный для выполнения функций хоста для CrimeListFragment. Начнем с создания представления для CrimeListActivity.

Обобщенный макет для хостинга фрагмента

Для CrimeListActivity можно просто воспользоваться макетом, определенным в файле activity_crime.xml (листинг 9.7). Этот макет определяет виджет FrameLayout как контейнерное представление для фрагмента, который затем указывается в коде активности.

Листинг 9.7. Файл activity_crime.xml уже содержит универсальную разметку

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragmentContainer"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  />
```

Поскольку в файле activity_crime.xml не указан конкретный фрагмент, он может использоваться для любой активности, выполняющей функции хоста для одного фрагмента. Переименуем его в activity_fragment.xml, чтобы отразить этот факт.

Закройте файл activity_crime.xml в редакторе (если он открыт). Затем на панели Package Explorer щелкните правой кнопкой мыши на файле res/layout/activity_crime.xml. (Будьте внимательны — щелкнуть нужно на activity_crime.xml, а не на fragment_crime.xml.)

Выберите в контекстном меню команду Refactor ▸ Rename... Введите имя activity_fragment.xml. При переименовании ресурса ссылки на него обновляются автоматически.

В старых версиях ADT переименование ресурса не сопровождалось обновлением ссылок. Если Eclipse выдает сообщение об ошибке в CrimeActivity.java, вам придется вручную обновить ссылку CrimeActivity, как показано в листинге 9.8.

Листинг 9.8. Обновление файла макета для CrimeActivity (CrimeActivity.java)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при исходном создании активности. */
```

продолжение ⇨

Листинг 9.8 (*продолжение*)

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

Абстрактный класс Activity

Для создания класса CrimeListActivity можно повторно использовать код CrimeActivity. Взгляните на код, написанный для CrimeActivity (листинг 9.8): он прост и практически универсален. Собственно, в нем есть всего одно не-универсальное место: создание экземпляра CrimeFragment перед его добавлением в FragmentManager.

Листинг 9.9. Класс CrimeActivity почти универсален (CrimeActivity.java)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

Почти в каждой активности, которая будет создаваться в этой книге, будет присутствовать такой же код. Чтобы нам не приходилось вводить его снова и снова, мы выделим его в абстрактный класс.

Создайте новый класс с именем SingleFragmentActivity в пакете CriminalIntent. Сделайте его субклассом FragmentActivity и установите флажок abstract, чтобы сделать SingleFragmentActivity абстрактным классом (рис. 9.3).

Рис. 9.3. Создание абстрактного класса SingleFragmentActivity

Щелкните на кнопке Finish и включите следующий фрагмент в SingleFragmentActivity.java. Не считая выделенных частей, он идентичен старому коду CrimeActivity.

Листинг 9.10. Добавление обобщенного суперкласса (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);
        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

В этом коде представление активности заполняется по данным activity_fragment.xml. Затем мы ищем фрагмент в FragmentManager этого контейнера, создавая и добавляя его, если он не существует.

Код в листинге 9.10 отличается от кода CrimeActivity только абстрактным методом createFragment(), который используется для создания экземпляра фрагмента. Субклассы SingleFragmentActivity реализуют этот метод так, чтобы возвращал экземпляр фрагмента, хостом которого является активность.

Использование абстрактного класса

Попробуем использовать класс с CrimeListActivity. Создайте новый класс с именем CrimeListActivity. Назначьте SingleFragmentActivity его суперклассом в мастере. Щелкните на кнопке Browse, введите имя SingleFragmentActivity, и Eclipse предложит его в списке вариантов. Выберите его и нажмите кнопку Finish.

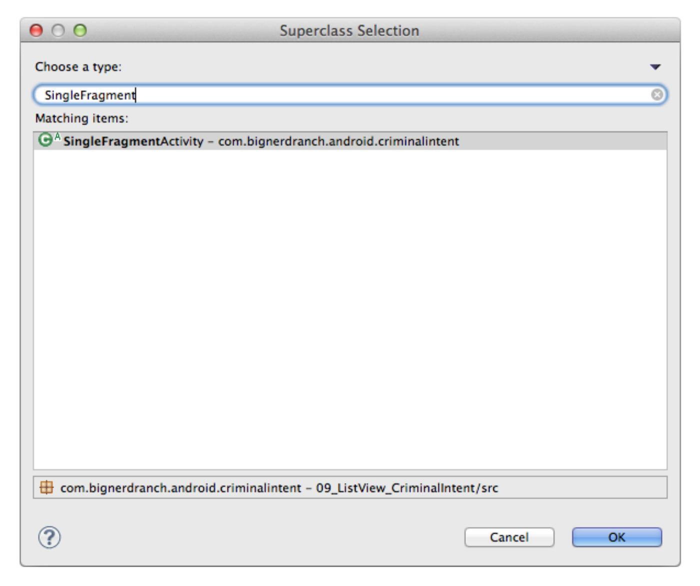


Рис. 9.4. Выбор SingleFragmentActivity

Eclipse открывает файл CrimeListActivity.java, в котором уже присутствует заготовка для createFragment(). Метод возвращает новый экземпляр CrimeListFragment.

Листинг 9.11. Реализация CrimeListActivity (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

}
```

Было бы лучше, если бы класс CrimeActivity работал аналогичным образом. Вернитесь к файлу CrimeActivity.java. Удалите существующий код из CrimeActivity и преобразуйте его в субкласс SingleFragmentActivity, как показано в листинге 9.12.

Листинг 9.12. Переработка CrimeListActivity (CrimeListActivity.java)

```
public class CrimeActivity extends FragmentActivity SingleFragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}
```

Класс SingleFragmentActivity избавляет от ввода лишнего текста и сэкономит немало времени в примерах книги. Кроме того, с ним код активности становится более компактным и аккуратным.

Объявление CrimeListActivity

Теперь, когда класс CrimeListActivity создан, его следует объявить в манифесте. Кроме того, список преступлений должен выводиться на первом экране, который виден пользователю после запуска CriminalIntent; следовательно, активность CrimeListActivity должна быть активностью лаунчера.

Включите в манифест объявление CrimeListActivity и переместите фильтр интентов из объявления CrimeActivity в объявление CrimeListActivity, как показано в листинге 9.13.

Листинг 9.13. Объявление CrimeListActivity активностью лаунчера (AndroidManifest.xml)

```
  ...

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity android:name=".CrimeListActivity">
```

продолжение ⇧

Листинг 9.13 (*продолжение*)

```
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".CrimeActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>

</manifest>
```

`CrimeListActivity` теперь является активностью лаунчера. Запустите CriminalIntent; на экране появляется виджет `FrameLayout` из `CrimeListActivity`, содержащий пустой фрагмент `CrimeListFragment`.

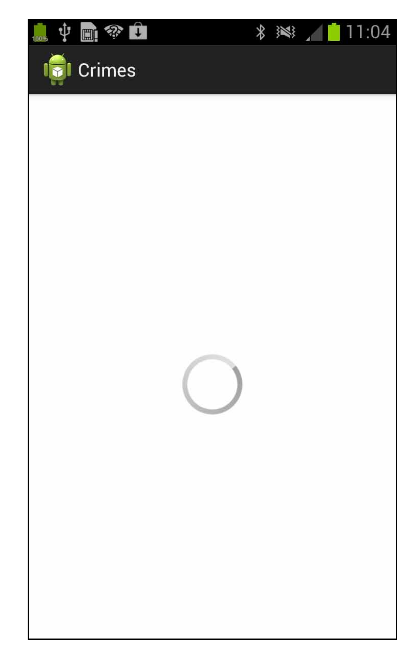


Рис. 9.5. Пустой экран CrimeListActivity

Когда у `ListView` нет данных для отображения, `ListFragment` выводит круговой индикатор прогресса. Мы предоставили `CrimeListFragment` доступ к массиву объектов `Crime`, но еще не сделали ничего, чтобы вывести их данные в `ListView`. Это станет нашей следующей задачей.

ListFragment, ListView и ArrayAdapter

Вместо кругового индикатора в виджете ListView из CrimeListFragment должен отображаться список. Каждый элемент этого списка содержит данные об одном экземпляре Crime.

ListView является субклассом ViewGroup, а каждый элемент списка отображается как дочерний объект View виджета ListView. В зависимости от сложности отображаемых данных дочерние объекты View могут быть сложными или очень простыми.

Наша первая реализация передачи данных для отображения будет очень простой: в элементе списка будет отображаться только краткое описание объекта Crime, а объект View представляет собой простой виджет TextView.



Рис. 9.6. ListView с дочерними виджетами TextView

На рис. 9.6 видны 11 виджетов TextView и часть 12-го. Если бы этот список можно было прокрутить, в виджете ListView отобразились бы дополнительные виджеты TextView — Crime #12, Crime #13 и т. д.

Откуда берутся эти объекты View? Может, полный список создается в ListView заранее? Это было бы неэффективно. Объект View должен существовать только тогда, когда он непосредственно находится на экране. Списки бывают огромными, создание и хранение объектов представления для всего списка создаст проблемы с быстродействием и памятью.

Разумнее создавать объекты представлений только тогда, когда в них возникнет необходимость. `ListView` запрашивает объект представления тогда, когда потребуется вывести определенный элемент списка.

К кому `ListView` обращается с этим запросом? К своему *адаптеру* (adapter) — объекту контроллера, который находится между `ListView` и набором данных с информацией, которую должен вывести `ListView`.

Адаптер отвечает за:

- создание необходимого объекта представления;
- заполнение его данными из уровня модели;
- возвращение объекта представления `ListView`.

Адаптер представляет собой экземпляр класса, реализующего интерфейс `Adapter`. Мы будем использовать экземпляр `ArrayAdapter<T>` — адаптера, который умеет работать с данными массива (или `ArrayList`) объектов типа `T`.

На рис. 9.7 представлена «родословная» класса `ArrayAdapter<T>`. Каждое звено цепи обеспечивает дополнительный уровень специализации.

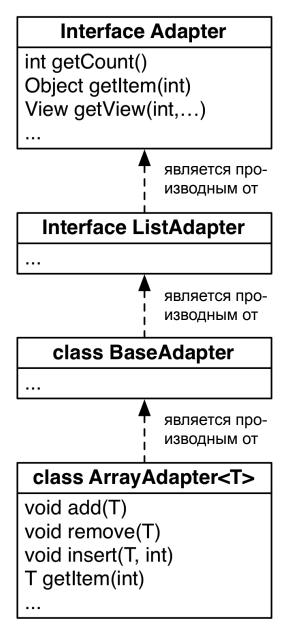


Рис. 9.7. Иерархия наследования ArrayAdapter<T>

Когда виджету `ListView` требуется объект представления для отображения, он вступает в диалог со своим адаптером. На рис. 9.8 приведен пример возможного диалога `ListView` с адаптером массива.

Сначала `ListView` запрашивает общее количество объектов в массиве, для чего он вызывает метод `getCount()` адаптера (это необходимо для того, чтобы избежать ошибок выхода за границу массива).

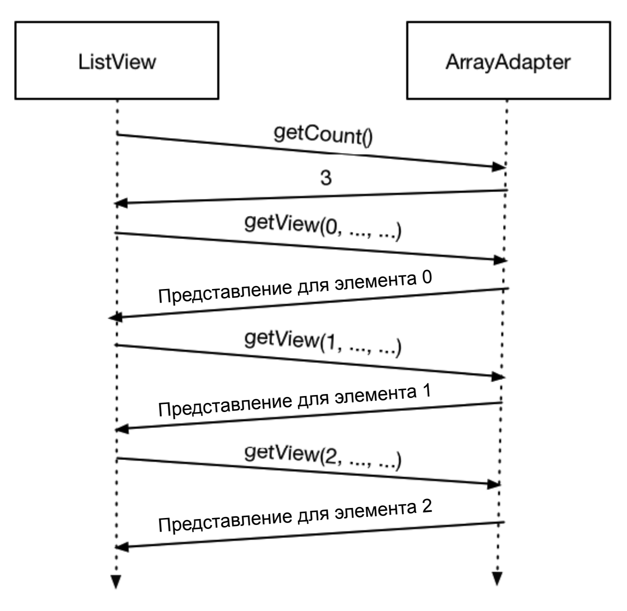


Рис. 9.8. Диалог ListView-Adapter

Затем `ListView` вызывает метод `getView(int, View, ViewGroup)` адаптера. В первом аргументе передается позиция элемента списка, который нужен `ListView`.

В своей реализации `getView(…)` адаптер создает объект представления по указанному элементу массива и возвращает этот объект представления объекту `ListView`. Последний включает объект представления в себя как дочернее представление, в результате чего новое представление оказывается на экране.

Механика `getView(…)` будет подробно описана далее в этой главе, когда мы переопределим этот метод для создания нестандартных элементов списка.

Создание ArrayAdapter<T>

Начнем с создания реализации `ArrayAdapter<T>` по умолчанию для `CrimeListFragment` с использованием следующего конструктора:

```
public ArrayAdapter(Context context, int textViewResourceId, T[] objects)
```

В конструкторе адаптера массива первым параметром является объект `Context`, необходимый для использования идентификатора ресурса из второго параметра. Идентификатор ресурса определяет макет, который будет использоваться `ArrayAdapter` для создания объекта представления. В третьем параметре передается набор данных.

В файле CrimeListFragment.java создайте экземпляр `ArrayAdapter<T>` и назначьте его адаптером для виджета `ListView` из `CrimeListFragment` (листинг 9.14).

Листинг 9.14. Создание ArrayAdapter (CrimeListFragment.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getActivity().setTitle(R.string.crimes_title);
    mCrimes = CrimeLab.get(getActivity()).getCrimes();

    ArrayAdapter<Crime> adapter =
       new ArrayAdapter<Crime>(getActivity(),
                               android.R.layout.simple_list_item_1,
                               mCrimes);
    setListAdapter(adapter);
}
```

`setListAdapter(ListAdapter)` — вспомогательный метод `ListFragment`, который может использоваться для назначения адаптера объекта `ListView`, находящегося под управлением `CrimeListFragment`.

Макет, заданный в конструкторе адаптера (`android.R.layout.simple_list_item_1`), представляет собой заранее определенный макет из ресурсов, предоставляемых Android SDK. Корневым элементом этого макета является элемент `TextView`.

Листинг 9.15. Исходный код android.R.layout.simple_list_item_1

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@android:id/text1"
  style="?android:attr/listItemFirstLineStyle"
  android:paddingTop="2dip"
  android:paddingBottom="3dip"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" />
```

В этом конструкторе также можно задать другой макет — при условии, что его корневым элементом является `TextView`.

Благодаря поведению по умолчанию, реализованному `ListFragment`, вы теперь можете запустить приложение. Экземпляр `ListView` будет успешно создан, появится на экране и начнет взаимодействовать со своим адаптером.

Запустите CriminalIntent. Вместо кругового индикатора в списке будут отображаться элементы. Впрочем, текст, выводимый в каждом представлении, не особенно полезен для пользователя.

Реализация `ArrayAdapter<T>.getView(…)` по умолчанию базируется на `toString()`. Она заполняет макет, находит нужный объект `Crime`, а затем вызывает `toString()` для объекта, чтобы заполнить `TextView`.

`Crime` пока не переопределяет `toString()`, поэтому используется реализация класса `java.lang.Object`, которая возвращает полное имя класса и адрес памяти объекта.

Чтобы адаптер создавал более содержательное представление для `Crime`, откройте файл Crime.java и переопределите метод `toString()` так, чтобы он возвращал краткое описание преступления.

Листинг 9.16. Переопределение Crime.toString() (Crime.java)

```
...
public Crime() {
    mId = UUID.randomUUID();
    mDate = new Date();
}

@Override
public String toString() {
    return mTitle;
}

...
```

Снова запустите CriminalIntent. Прокрутите список и просмотрите его элементы.



Рис. 9.9. Элементы списка с именем класса и адресами памяти



Рис. 9.10. Простые элементы списка с краткими описаниями преступлений

В процессе прокрутки `ListView` вызывает метод `getView(…)` адаптера для получения представлений, которые требуется отобразить.

Щелчки на элементах списка

Чтобы отреагировать на прикосновение пользователя к элементу списка, следует переопределить другой вспомогательный метод `ListFragment`:

```
public void onListItemClick(ListView l, View v, int position, long id)
```

Независимо от способа выполнения щелчка: физической кнопкой, программной кнопкой, касанием пальца — результат все равно проходит через `onListItemClick(...)`.

В файле CrimeListFragment.java переопределите `onListItemClick(...)`. Адаптер должен возвращать объект `Crime` для элемента списка, на котором был сделан щелчок, и регистрировать краткое описание этого объекта `Crime`.

Листинг 9.17. Переопределение метода onListItemClick(...) для регистрации кратких описаний Crime (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {
    private static final String TAG = "CrimeListFragment";

    ...

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        Crime c = (Crime)(getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
    }
```

`getListAdapter()` — вспомогательный метод `ListFragment`, возвращающий адаптер, который назначается представлению списка объекта `ListFragment`. Затем мы вызываем метод `getItem(int)` адаптера с использованием параметра `position` метода `onListItemClick(...)` и преобразуем результат в `Crime`.

Снова запустите CriminalIntent. Щелкните на элементе списка и убедитесь в том, что в журнале зарегистрирован правильный объект `Crime`.

Настройка элементов списка

До настоящего момента в каждом элементе списка выводится только краткое описание `Crime` — результат вызова `Crime.toString()`.

Если вы хотите, чтобы элементы списка не ограничивались выводом простого текста, создайте пользовательский элемент списка. Для реализации пользовательских элементов необходимо решить две задачи:

- создать в XML новый макет, определяющий представление элемента списка;
- создать субкласс `ArrayAdapter<T>`, который умеет создавать, заполнять и возвращать представление, определенное в новом макете.

Создание макета элемента списка

В приложении CriminalIntent макет элемента списка должен включать краткое описание преступления, дату и признак раскрытия (рис. 9.11). Такой макет состоит из двух виджетов `TextView` и `CheckBox`.

Новый макет создается точно так же, как для представления активности или фрагмента. На панели Package Explorer щелкните правой кнопкой мыши на каталоге res/layout и выберите команду New ▸ Other... ▸ Android XML File. В открывшемся диалоговом окне выберите тип ресурса Layout, введите имя файла list_item_crime.xml, выберите корневой элемент RelativeLayout и щелкните на кнопке Finish.

RelativeLayout позволяет использовать параметры макета для размещения дочерних представлений относительно корневого макета и друг друга. Мы хотим, чтобы виджет CheckBox автоматически выравнивался по правой стороне RelativeLayout. Два виджета TextView будут выравниваться относительно CheckBox.



Рис. 9.11. Список с пользовательским форматом элементов

На рис. 9.12 изображены виджеты макета пользовательского элемента списка. Виджет CheckBox должен определяться первым несмотря на то, что он находится у правого края макета. Это связано с тем, что TextView будут использовать идентификатор CheckBox в значении атрибута.

RelativeLayout
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"

CheckBox
android:id="**@+id/crime_list_item_solvedCheckBox**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:gravity="**center**"
android:layout_alignParentRight="**true**"
android:enabled="**false**"
android:padding="**4dp**"

layout_below

TextView
android:id="**@+id/crime_list_item_titleTextView**"
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"
android:layout_toLeftOf="**@id/crime_list_item_solvedCheckBox**"
android:textStyle="**bold**"
android:paddingLeft="**4dp**"
android:paddingRight="**4dp**"
android:text="**Crime title**"

layout_below

layout_toLeftOf

TextView
android:id="**@+id/crime_list_item_dateTextView**"
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"
android:layout_below="**@id/crime_list_item_titleTextView**"
android:layout_toLeftOf="**@id/crime_list_item_solvedCheckBox**"
android:paddingLeft="**4dp**"
android:paddingRight="**4dp**"
android:paddingTop="**4dp**"
android:text="**Crime date**"

Рис. 9.12. Макет пользовательского элемента списка (list_item_crime.xml)

По той же причине определение виджета TextView с кратким описанием должно предшествовать определению виджета TextView с датой. В файле макета виджет должен определяться до того, как другие виджеты смогут использовать его идентификатор в своих определениях.

Обратите внимание: при использовании идентификатора виджета в определении другого виджета не нужно включать знак +. Он используется для создания идентификатора при его первом вхождении в файл макета — обычно в атрибуте android:id.

Также обратите внимание на использование строковых литералов вместо строковых ресурсов в атрибутах android:text. Эти текстовые атрибуты содержат фиктивный текст для разработки и тестирования. Значения, видимые пользователю, предоставляет адаптер. Поскольку эти конкретные строки никогда не будут видны пользователю, создавать для них строковые ресурсы бессмысленно.

Макет пользовательского элемента списка завершен, и мы можем переходить к следующему шагу — созданию пользовательского адаптера.

Создание субкласса адаптера

Наш макет предназначен для отображения элементов списка, представляющих объекты Crime. Для получения данных этих элементов используются методы доступа Crime, так что нам понадобится новый адаптер, умеющий работать с объектами Crime.

В файле CrimeListFragment.java создайте субкласс ArrayAdapter как внутренний класс CrimeListFragment.

Листинг 9.18. Добавление пользовательского адаптера как внутреннего класса (CrimeListFragment.java)

```
    public void onListItemClick(ListView l, View v, int position, long id) {
        Crime c = (Crime)(getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
    }

    private class CrimeAdapter extends ArrayAdapter<Crime> {

        public CrimeAdapter(ArrayList<Crime> crimes) {
            super(getActivity(), 0, crimes);
        }
    }
}
```

Вызов конструктора суперкласса необходим для подключения набора данных Crime. Мы не будем использовать предопределенный макет, поэтому вместо идентификатора макета можно передать 0.

Пользовательский элемент списка создается и возвращается в методе ArrayAdapter<T>:

```
public View getView(int position, View convertView, ViewGroup parent)
```

Параметр convertView содержит существующий элемент списка, который может быть изменен и возвращен адаптером вместо создания нового объекта. Повторное

создание объектов представления повышает быстродействие приложения, потому что оно позволяет избежать постоянного создания и уничтожения однотипных объектов. Количество элементов, одновременно отображаемых в ListView, ограничено, поэтому было бы неэффективно содержать множество неиспользуемых объектов представлений, попусту расходующих память.

В классе CrimeAdapter переопределите метод getView(…). Метод должен возвращать представление, полученное заполнением пользовательского макета и содержащее правильные данные Crime (листинг 9.19).

Листинг 9.19. Переопределение getView(…) (CrimeListFragment.java)

```
private class CrimeAdapter extends ArrayAdapter<Crime> {
    public CrimeAdapter(ArrayList<Crime> crimes) {
        super(getActivity(), 0, crimes);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Если мы не получили представление, заполняем его
        if (convertView == null) {
            convertView = getActivity().getLayoutInflater()
                .inflate(R.layout.list_item_crime, null);
        }

        // Настройка представления для объекта Crime
        Crime c = getItem(position);

        TextView titleTextView =
            (TextView)convertView.findViewById(R.id.crime_list_item_titleTextView);
        titleTextView.setText(c.getTitle());
        TextView dateTextView =
            (TextView)convertView.findViewById(R.id.crime_list_item_dateTextView);
        dateTextView.setText(c.getDate().toString());
        CheckBox solvedCheckBox =
            (CheckBox)convertView.findViewById(R.id.crime_list_item_solvedCheckBox);
        solvedCheckBox.setChecked(c.isSolved());

        return convertView;
    }
}
```

В этой реализации getView(…) мы сначала проверяем, был ли передан существующий объект, предназначенный для переработки. Если нет — объект заполняется по текущему макету.

Как с новым, так и с перерабатываемым объектом вызывается метод getItem(int) класса Adapter для получения объекта Crime в текущей позиции списка.

После получения нужного объекта Crime мы получаем ссылку на каждый виджет в объекте представления и настраиваем его данными Crime. В конце объект представления возвращается ListView.

Теперь мы можем подключить свой адаптер к CrimeListFragment. В начале файла CrimeListFragment.java обновите реализации onCreate(…) и onListItemClick(…) для использования CrimeAdapter, как показано в листинге 9.20.

Листинг 9.20. Использование CrimeAdapter (CrimeListFragment.java)

```
    ArrayAdapter<Crime> adapter = new ArrayAdapter<Crime>(this,
            android.R.layout.simple_list_item_1,
            mCrimes);
    CrimeAdapter adapter = new CrimeAdapter(mCrimes);
    setListAdapter(adapter);
}

public void onListItemClick(ListView l, View v, int position, long id) {
    Crime c = (Crime)(getListAdapter()).getItem(position);
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    Log.d(TAG, c.getTitle() + " was clicked");
}
```

Приведение к типу `CrimeAdapter` позволяет использовать преимущества проверки типов. `CrimeAdapter` может содержать только объекты `Crime`, поэтому преобразование к `Crime` становится лишним.

В обычной ситуации приложение было бы готово к запуску. Однако из-за того, что элемент списка содержит `CheckBox`, необходимо внести еще одно изменение. Виджет `CheckBox` по умолчанию может получать фокус ввода. Соответственно щелчок на элементе списка будет интерпретироваться как переключение состояния `CheckBox` и не достигнет метода `onListItemClick(…)`.

Рис. 9.13. Список с пользовательскими элементами

Эта внутренняя особенность ListView означает, что для любого виджета с поддержкой фокуса, присутствующего в макете элемента списка (например, CheckBox или Button), получение фокуса следует запретить. Это необходимо для того, чтобы щелчки на элементах списка работали так, как вы ожидаете.

Поскольку элемент CheckBox используется только для вывода информации и не связывается с логикой приложения, существует простое решение: достаточно внести изменение в файл list_item_crime.xml и запретить передачу фокуса CheckBox.

Листинг 9.21. Запрет передачи фокуса CheckBox (list_item_crime.xml)

```
...
<CheckBox android:id="@+id/crime_list_item_solvedCheckBox"
  android:layout_width="wrap_content"
  android:layout_height="match_parent"
  android:gravity="center"
  android:layout_alignParentRight="true"
  android:enabled="false"
  android:focusable="false"
  android:padding="4dp" />
...
```

Запустите приложение и прокрутите список. Щелкните на элементе списка и по журналу убедитесь в том, что CrimeAdapter возвращает правильные данные. Если приложение запустилось, но макет выглядит неправильно, вернитесь к файлу list_item_crime.xml и проверьте определение макета.

10 Аргументы фрагментов

В этой главе мы наладим совместную работу списка и детализации в приложении CriminalIntent. Когда пользователь щелкает на элементе списка преступлений, на экране появляется новый экземпляр `CrimeActivity`, который является хостом для экземпляра `CrimeFragment` с подробной информацией о конкретном экземпляре `Crime`.

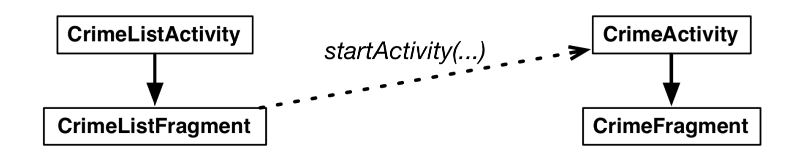


Рис. 10.1. Запуск CrimeActivity из CrimeListActivity

В приложении GeoQuiz одна активность (`QuizActivity`) запускала другую (`CheatActivity`). В приложении CriminalIntent активность `CrimeActivity` будет запускаться из фрагмента `CrimeListFragment`.

Запуск активности из фрагмента

Запуск активности из фрагмента осуществляется практически так же, как запуск активности из другой активности. Вы вызываете метод `Fragment.startActivity(Intent)`, который вызывает соответствующий метод `Activity` во внутренней реализации.

В реализации `onListItemClick(…)` из `CrimeListFragment` замените регистрацию краткого описания кодом, запускающим экземпляр `CrimeActivity`. (Не обращайте внимания на предупреждение о неиспользуемой переменной `Crime`; мы используем ее в следующем разделе.)

Листинг 10.1. Запуск CrimeActivity (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Получение объекта Crime от адаптера
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    Log.d(TAG, c.getTitle() + " was clicked");

    // Запуск CrimeActivity
    Intent i = new Intent(getActivity(), CrimeActivity.class);
    startActivity(i);
}
```

Здесь класс `CrimeListFragment` создает явный интент с указанием класса `CrimeActivity`. `CrimeListFragment` использует метод `getActivity()` для передачи активности-хоста как объекта `Context`, необходимого конструктору `Intent`.

Запустите приложение CriminalIntent. Щелкните на любой строке списка; открывается новый экземпляр `CrimeActivity`, управляющий фрагментом `CrimeFragment`.

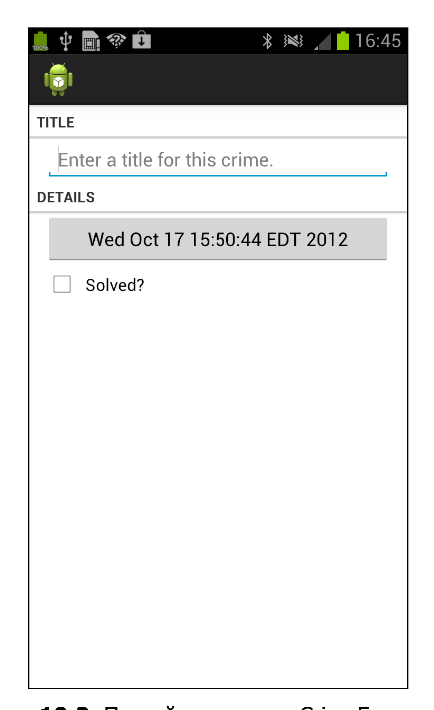


Рис. 10.2. Пустой экземпляр CrimeFragment

Экземпляр `CrimeFragment` еще не содержит данных конкретного объекта `Crime`, потому что мы не сообщили ему, какой именно объект `Crime` следует отображать.

Включение дополнения

Чтобы сообщить `CrimeFragment`, какой объект `Crime` следует отображать, можно сделать `mCrimeId` дополнением (extra) объекта `Intent`. В методе `onListItemClick(...)`

включите поле mCrimeId выбранного объекта Crime в интент, запускающий CrimeActivity.

В следующем коде Eclipse обнаруживает ошибку — ключ CrimeFragment.EXTRA_CRIME_ID еще не создан. Не обращайте внимания на ошибку, вскоре мы создадим его.

Листинг 10.2. Запуск CrimeActivity с дополнением (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Получение объекта Crime от адаптера
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);

    // Запуск CrimeActivity
    Intent i = new Intent(getActivity(), CrimeActivity.class);
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
    startActivity(i);
}
```

После создания явного интента мы вызываем putExtra(…), передавая строковый ключ и связанное с ним значение (mCrimeId). В данном случае вызывается версия putExtra(String, Serializable), потому что UUID является объектом Serializable.

Чтение дополнения

Поле mCrimeId сохранено в интенте, принадлежащем CrimeActivity, однако прочитать и использовать эти данные должен класс CrimeFragment.

Существует два способа, которыми фрагмент может обратиться к данным из интента активности: простое и прямолинейное обходное решение и сложная, гибкая полноценная реализация. Сначала мы опробуем первый способ, а потом реализуем сложное гибкое решение с *аргументами фрагментов*.

В простом решении CrimeFragment просто использует метод getActivity() для прямого обращения к интенту CrimeActivity. Вернитесь к классу CrimeFragment и добавьте ключ для дополнения, затем в методе onCreate(Bundle) прочитайте дополнительный из интента CrimeActivity и используйте его для получения данных Crime.

Листинг 10.3. Получение дополнения и выборка Crime (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private Crime mCrime;

    ...

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        UUID crimeId = (UUID)getActivity().getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);

        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}
```

Если не считать вызова `getActivity()`, листинг 10.3 практически не отличается от кода выборки дополнения из кода активности. Метод `getIntent()` возвращает объект `Intent`, используемый для запуска `CrimeActivity`. Мы вызываем `getSerializableExtra(String)` для `Intent`, чтобы извлечь UUID в переменную.

После получения идентификатора мы используем его для получения объекта `Crime` от `CrimeLab`. Методу `CrimeLab.get(…)` должен передаваться объект `Context`, поэтому `CrimeFragment` передает `CrimeActivity`.

Обновление представления CrimeFragment данными Crime

Теперь, когда фрагмент `CrimeFragment` получает объект `Crime`, его представление может отобразить данные `Crime`. Обновите метод `onCreateView(…)`, чтобы он выводил краткое описание преступления и признак раскрытия (код вывода даты уже имеется).

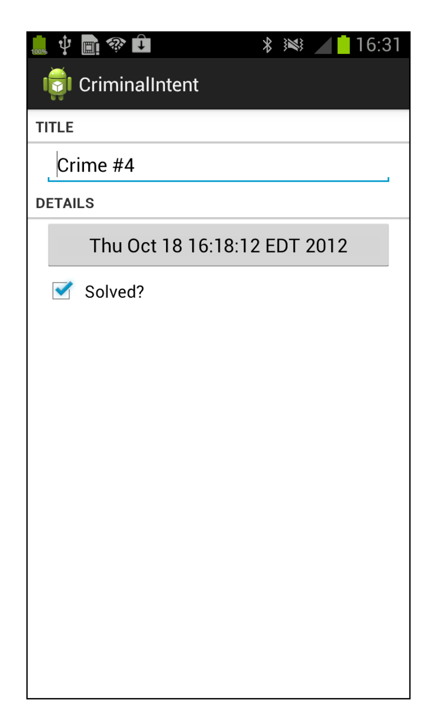


Рис. 10.3. Преступление, выбранное в списке

Листинг 10.4. Обновление объектов представления (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...

    mTitleField = (EditText)v.findViewById(R.id.crime_title);
```

продолжение ⇗

Листинг 10.4 (*продолжение*)

```
        mTitleField.setText(mCrime.getTitle());
        mTitleField.addTextChangedListener(new TextWatcher() {
            ...
        });
        ...
        mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
        mSolvedCheckBox.setChecked(mCrime.isSolved());
        mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            ...
        });
        ...

        return v;
    }
```

Запустите приложение CriminalIntent. Выберите строку Crime #4 и убедитесь в том, что на экране появляется экземпляр CrimeFragment с правильными данными преступления.

Недостаток прямой выборки

Обращение из фрагмента к интенту, принадлежащему активности-хосту, упрощает код. С другой стороны, оно нарушает инкапсуляцию фрагмента. Класс CrimeFragment уже не является структурным элементом, пригодным для повторного использования, потому что он предполагает, что его хостом всегда будет активность с объектом Intent, определяющим дополнение с именем EXTRA_CRIME_ID.

Возможно, для CrimeFragment такое предположение разумно, но оно означает, что класс CrimeFragment в своей текущей реализации не может использоваться с произвольной активностью.

Другое, более правильное решение — сохранение значения mCrimeId в месте, принадлежащем CrimeFragment (вместо хранения его в личном пространстве CrimeActivity). В этом случае объект CrimeFragment может прочитать данные, не полагаясь на присутствие конкретного дополнения в интенте активности. Такое «место», принадлежащее фрагменту, называется *пакетом аргументов* (arguments bundle).

Аргументы фрагментов

К каждому экземпляру фрагмента может быть прикреплен объект Bundle. Этот объект содержит пары «ключ-значение», которые работают так же, как дополнения интентов Activity. Каждая такая пара называется *аргументом* (argument).

Чтобы создать аргументы фрагментов, вы сначала создаете объект Bundle, а затем используете put-методы Bundle соответствующего типа (по аналогии с методами Intent) для добавления аргументов в пакет.

```
Bundle args = new Bundle();
args.putSerializable(EXTRA_MY_OBJECT, myObject);
args.putInt(EXTRA_MY_INT, myInt);
args.putCharSequence(EXTRA_MY_STRING, myString);
```

Присоединение аргументов к фрагменту

Чтобы присоединить пакет аргументов к фрагменту, вызовите метод Fragment.setArguments(Bundle). Присоединение должно быть выполнено после создания фрагмента, но до его добавления в активность.

Для этого программисты Android используют схему с добавлением в класс Fragment статического метода с именем newInstance(). Этот метод создает экземпляр фрагмента, упаковывает и задает его аргументы.

Когда активности-хосту потребуется экземпляр этого фрагмента, она вместо прямого вызова конструктора вызывает метод newInstance(). Активность может передать newInstance(…) любые параметры, необходимые фрагменту для создания аргументов.

Включите в CrimeFragment метод newInstance(UUID), который получает UUID, создает пакет аргументов, создает экземпляр фрагмента, а затем присоединяет аргументы к фрагменту.

Листинг 10.5. Метод newInstance(UUID) (CrimeFragment.java)

```
public static CrimeFragment newInstance(UUID crimeId) {
    Bundle args = new Bundle();
    args.putSerializable(EXTRA_CRIME_ID, crimeId);
    CrimeFragment fragment = new CrimeFragment();
    fragment.setArguments(args);
    return fragment;
}
```

Теперь класс CrimeActivity должен вызывать CrimeFragment.newInstance(UUID) каждый раз, когда ему потребуется создать CrimeFragment. При вызове передается значение UUID, полученное из дополнения. Вернитесь к классу CrimeActivity, в методе createFragment() получите дополнение из интента CrimeActivity и передайте его CrimeFragment.newInstance(UUID).

Листинг 10.6. Использование newInstance(UUID) (CrimeActivity.java)

```
@Override
protected Fragment createFragment() {
    return new CrimeFragment();

    UUID crimeId = (UUID)getIntent()
        .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);

    return CrimeFragment.newInstance(crimeId);
}
```

Учтите, что потребность в независимости не является двусторонней. Класс CrimeActivity должен многое знать о классе CrimeFragment — например, то, что он содержит метод newInstance(UUID). Это нормально; активность-хост должна располагать конкретной информацией о том, как управлять фрагментами, но фрагментам такая информация об их активности не нужна (по крайней мере если вы хотите сохранить гибкость независимых фрагментов).

Получение аргументов

Когда фрагменту требуется получить доступ к его аргументам, он вызывает метод getArguments() класса Fragment, а затем один из get-методов Bundle для конкретного типа.

В методе CrimeFragment.onCreate(…) замените код упрощенного решения выборкой UUID из аргументов фрагмента.

Листинг 10.7. Получение идентификатора преступления из аргументов (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    UUID crimeId = (UUID)getActivity().getIntent()
        .getSerializableExtra(EXTRA_CRIME_ID);

    UUID crimeId = (UUID)getArguments().getSerializable(EXTRA_CRIME_ID);

    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}
```

Запустите приложение CriminalIntent. Оно работает точно так же, но архитектура с независимостью CrimeFragment должна вызвать у вас приятные чувства. Кроме того, она хорошо подготовит нас к следующей главе, в которой мы реализуем более сложную систему навигации в CriminalIntent.

Перезагрузка списка

Осталась еще одна подробность, которой нужно уделить внимание. Запустите приложение CriminalIntent, щелкните на элементе списка и внесите изменения в подробную информацию о преступлении. Эти изменения сохраняются в модели, но при возвращении к списку представление остается неизменным.

Мы должны сообщить адаптеру спискового представления, что набор данных изменился (или мог измениться), чтобы тот мог заново получить данные и повторно загрузить список. Работая со стеком возврата ActivityManager, можно перезагрузить список в нужный момент.

Когда CrimeListFragment запускает экземпляр CrimeActivity, последний помещается на вершину стека. При этом экземпляр CrimeActivity, который до этого находился на вершине, приостанавливается и останавливается.

Когда пользователь нажимает кнопку Back для возвращения к списку, экземпляр CrimeActivity извлекается из стека и уничтожается. В этот момент CrimeListActivity запускается и продолжает выполнение.

Когда экземпляр CrimeListActivity продолжает выполнение, он получает вызов onResume() от ОС. При получении этого вызова CrimeListActivity его экземпляр FragmentManager вызывает onResume() для фрагментов, хостом которых в настоящее

время является активность. В нашем случае это единственный фрагмент `CrimeList-Fragment`.

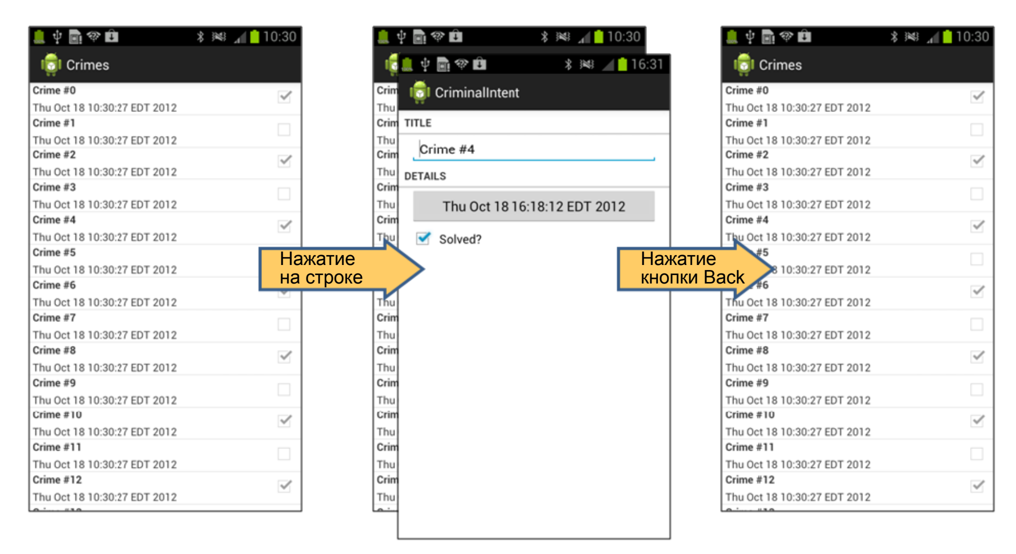


Рис. 10.4. Стек возврата CriminalIntent

В классе `CrimeListFragment` переопределите `onResume()` для перезагрузки списка.

Листинг 10.8. Перезагрузка списка в onResume() (CrimeListFragment.java)

```
@Override
public void onResume() {
    super.onResume();
    ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
}
```

Почему для обновления спискового представления переопределяется метод `onResume()`, а не `onStart()`? Мы не можем предполагать, что активность останавливается при нахождении перед ней другой активности. Если другая активность прозрачна, ваша активность может быть только приостановлена. Если же ваша активность приостановлена, а код обновления находится в `onStart()`, список не будет перезагружаться. В общем случае самым безопасным местом для выполнения действий, связанных с обновлением представления фрагмента, является метод `onResume()`.

Запустите приложение CriminalIntent. Выберите преступление в списке и измените его подробную информацию. Вернувшись к списку, вы немедленно увидите свои изменения.

За последние две главы приложение CriminalIntent серьезно продвинулось вперед. Давайте взглянем на обновленную диаграмму объектов.

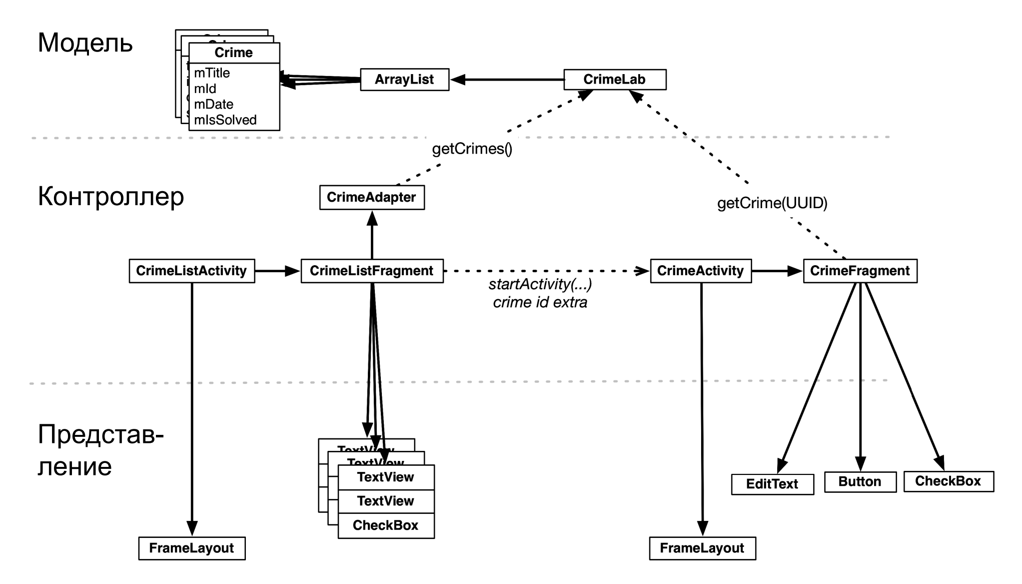


Рис. 10.5. Обновленная диаграмма объектов CriminalIntent

Получение результата с использованием фрагментов

В этой главе нам не требовалось, чтобы запущенная активность возвращала результат. А если бы это было нужно? Код выглядел бы почти так же, как в приложении GeoQuiz. Вместо метода `startActivityForResult(…)` класса `Activity` использовался бы метод `Fragment.startActivityForResult(…)`. Вместо `Activity.onActivityResult(…)` мы переопределим `Fragment.onActivityResult(…)`.

```
public class CrimeListFragment extends ListFragment {
    private static final int REQUEST_CRIME = 1;

    ...

    public void onListItemClick(ListView l, View v, int position, long id) {
        // Получение объекта Crime от адаптера
        Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
        // Запуск CrimeActivity
        Intent i = new Intent(getActivity(), CrimeActivity.class);
        startActivityForResult(i, REQUEST_CRIME);
    }
```

```
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CRIME) {
            // Обработка результата
        }
    }
}
```

Метод `Fragment.startActivityForResult(Intent,int)` похож на одноименный метод класса `Activity`. Он включает дополнительный код передачи результата вашему фрагменту от активности-хоста.

Возвращение результата от фрагмента выглядит немного иначе. Фрагмент может получить результат от активности, но не может вернуть собственный результат — на это способны только активности. Таким образом, хотя `Fragment` содержит методы `startActivityForResult(…)` и `onActivityResult(…)`, у него нет методов `setResult(…)`.

Вместо этого вы приказываете *активности-хосту* вернуть значение; вот как это делается:

```
public class CrimeFragment extends Fragment {
    ...

    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}
```

Возможность возвращения результата активности из фрагмента для приложения CriminalIntent продемонстрирована в главе 20.

11 ViewPager

В этой главе мы создадим новую активность, которая станет хостом для `CrimeFragment`. Макет активности будет состоять из экземпляра `ViewPager`. Включение виджета `ViewPager` в пользовательский интерфейс позволяет «листать» элементы списка, проводя пальцем по экрану.

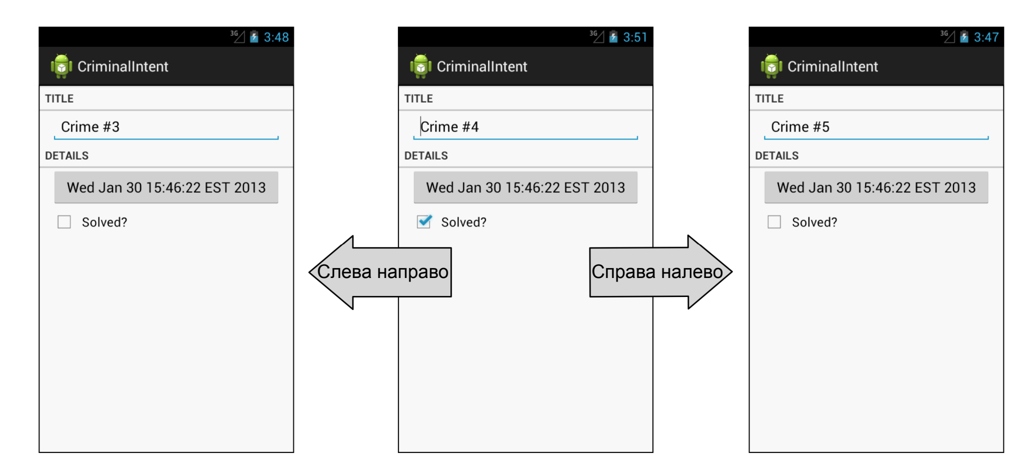


Рис. 11.1. Листание страниц

На рис. 11.2 представлена обновленная диаграмма CriminalIntent. Новая активность с именем `CrimePagerActivity` займет место `CrimeActivity`. Ее макет состоит из экземпляра `ViewPager`.

Все новые объекты, которые необходимо создать, находятся в пунктирном прямоугольнике на приведенной диаграмме. Для реализации листания страничных представлений в CriminalIntent ничего другого менять не придется. В частности, класс `CrimeFragment` останется неизменным благодаря той работе по обеспечению независимости `CrimeFragment`, которую мы проделали в главе 10.

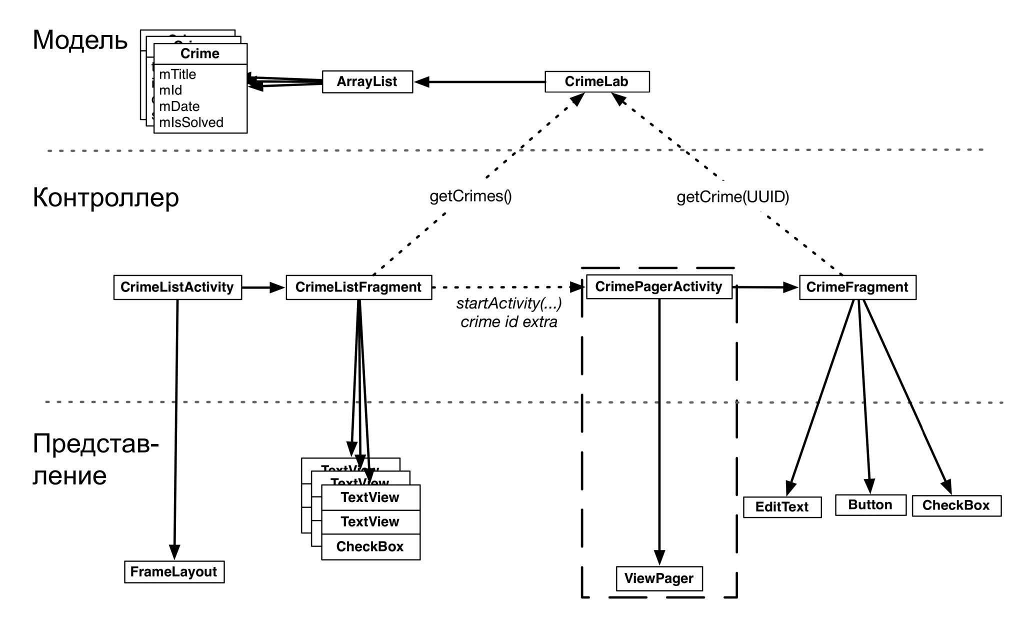


Рис. 11.2. Диаграмма макета CrimePagerActivity

В этой главе нам предстоит:

- создать класс `CrimePagerActivity`;
- определить иерархию представлений, состоящую из `ViewPager`;
- связать экземпляр `ViewPager` с его адаптером `CrimePagerActivity`;
- изменить метод `CrimeListFragment.onListItemClick(…)` так, чтобы он запускал `CrimePagerActivity` вместо `CrimeActivity`.

Создание CrimePagerActivity

Класс `CrimePagerActivity` будет субклассом `FragmentActivity`. Он создает и управляет экземпляром `ViewPager`.

Создайте новый класс с именем `CrimePagerActivity`. Назначьте его суперклассом `FragmentActivity`. В методе `onCreate(Bundle)` просто включите сквозной вызов суперкласса — создание экземпляра `ViewPager` будет рассмотрено чуть позднее.

Листинг 11.1. Создание ViewPager (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {
    private ViewPager mViewPager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Формирование макетов представлений в коде

В этой книге макеты представлений определяются исключительно в XML-файлах макетов. Обычно это хорошая идея, но в Android нет ничего, что бы заставляло вас определять макеты именно таким образом. Иерархия представлений в этой главе проста — она состоит из единственного представления. По этой причине вместо файла XML мы определим иерархию в программном коде. Так как в макете используется всего одно представление, эта задача будет решаться относительно просто.

Создание представления не требует никакого волшебства; просто вызовите его конструктор. К сожалению, полностью изолироваться от XML вам не удастся. Для некоторых компонентов все равно понадобятся идентификаторы ресурсов; `ViewPager` входит в их число. `FragmentManager` требует, чтобы любое представление, использованное как контейнер фрагмента, обязательно имело идентификатор.

`ViewPager` является контейнером фрагмента, поэтому ему необходимо присвоить идентификатор.

С учетом этого мы должны:

- создать идентификатор ресурса для `ViewPager`;
- создать экземпляр `ViewPager` и присвоить его `mViewPager`;
- настроить его, присвоив ему идентификатор ресурса;
- назначить `ViewPager` представлением содержимого активности.

Автономные идентификаторы ресурсов

Определение автономного идентификатора ресурса отчасти напоминает определение строкового ресурса: вы создаете элемент в файле XML из каталога res/values. Создайте новый файл ресурсов Android в формате XML с именем res/values/ids.xml для хранения идентификаторов и добавьте в него идентификатор с именем `viewPager`.

Листинг 11.2. Создание автономного идентификатора ресурса (res/values/ids.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">

    <item type="id" name="viewPager" />

</resources>
```

После того как идентификатор будет создан, можно переходить к созданию и отображению `ViewPager`. Создайте экземпляр `ViewPager` и назначьте его представлением содержимого.

Листинг 11.3. Программное создание представления содержимого (CrimePagerActivity.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mViewPager = new ViewPager(this);
    mViewPager.setId(R.id.viewPager);
    setContentView(mViewPager);
}
```

Класс `ViewPager` находится в библиотеке поддержки. В отличие от `Fragment` класс `ViewPager` доступен только в библиотеке поддержки; в последующих SDK не существует «стандартного» класса `ViewPager`.

ViewPager и PagerAdapter

Класс `ViewPager` в чем-то похож на `AdapterView` (суперкласс `ListView`). Чтобы класс `AdapterView` мог выдавать представления, ему необходим экземпляр `Adapter`. Классу `ViewPager` необходим адаптер `PagerAdapter`.

Однако взаимодействие между `ViewPager` и `PagerAdapter` намного сложнее взаимодействия между `AdapterView` и `Adapter`. К счастью, мы можем использовать `FragmentStatePagerAdapter` — субкласс `PagerAdapter`, который берет на себя многие технические подробности.

`FragmentStatePagerAdapter` сводит взаимодействие к двум простым методам: `getCount()` и `getItem(int)`. При вызове метода `getItem(int)` для позиции в массиве преступлений следует вернуть объект `CrimeFragment`, настроенный для вывода информации объекта в заданной позиции.

В классе `CrimePagerActivity` добавьте следующий код для назначения `PagerAdapter` класса `ViewPager` и реализации его методов `getCount()` и `getItem(int)`.

Листинг 11.4. Назначение PagerAdapter (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {
    private ViewPager mViewPager;
    private ArrayList<Crime> mCrimes;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mViewPager = new ViewPager(this);
        mViewPager.setId(R.id.viewPager);
        setContentView(mViewPager);

        mCrimes = CrimeLab.get(this).getCrimes();

        FragmentManager fm = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
            @Override
            public int getCount() {
                return mCrimes.size();
            }

            @Override
            public Fragment getItem(int pos) {
                Crime crime = mCrimes.get(pos);
                return CrimeFragment.newInstance(crime.getId());
            }
        });
    }
}
```

Пройдемся по этому коду. В первой строке мы получаем от CrimeLab набор данных — контейнер ArrayList объектов Crime. Затем мы получаем экземпляр FragmentManager для активности.

На следующем шаге адаптером назначается безымянный экземпляр FragmentStatePagerAdapter. Для создания FragmentStatePagerAdapter необходим объект FragmentManager. Не забывайте, что FragmentStatePagerAdapter — ваш агент, управляющий взаимодействием с ViewPager. Чтобы ваш агент мог выполнить свою работу с фрагментами, возвращаемыми в getItem(int), он должен быть способен добавить их в активность. Вот почему ему необходим экземпляр FragmentManager.

(Что именно делает агент? Вкратце он добавляет возвращаемые фрагменты в активность и помогает ViewPager идентифицировать представления фрагментов для их правильного размещения. Более подробная информация приведена в разделе «Для любознательных» в конце главы).

Два метода PagerAdapter весьма просты. Метод getCount() возвращает текущее количество элементов в списке. Все существенное происходит в методе getItem(int). Он получает экземпляр Crime для заданной позиции в наборе данных, после чего использует его идентификатор для создания и возвращения правильно настроенного экземпляра CrimeFragment.

Интеграция CrimePagerActivity

Теперь можно переходить к устранению класса CrimeActivity и замене его классом CrimePagerActivity.

Для начала щелчок на элементе списка в CrimeListFragment должен запускать экземпляр CrimePagerActivity вместо CrimeActivity.

Вернитесь к файлу CrimeListFragment.java и измените метод onListItemClick(…) так, чтобы он запускал CrimePagerActivity.

Листинг 11.5. Запуск активности (CrimeListFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Получение объекта Crime от адаптера
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    // Запуск CrimeActivity
    Intent i = new Intent(getActivity(), CrimeActivity.class);
    // Запуск CrimePagerActivity с объектом Crime
    Intent i = new Intent(getActivity(), CrimePagerActivity.class);
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
    startActivity(i);
}
```

Также необходимо добавить CrimePagerActivity в манифест, чтобы ОС могла запустить эту активность. Пока манифест будет открыт, заодно удалите объявление CrimeActivity.

Листинг 11.6. Добавление CrimePagerActivity в манифест (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  ...
  <application ...>
    ...
    <activity
      android:name=".CrimeActivity"
      android:label="@string/app_name" >
    </activity>
    <activity android:name=".CrimePagerActivity"
      android:label="@string/app_name">
    </activity>

  </application>

</manifest>
```

Наконец, чтобы не загромождать проект, удалите CrimeActivity.java на панели Package Explorer.

Запустите приложение CriminalIntent. Нажмите на строке Crime #0, чтобы просмотреть подробную информацию. Проведите по экрану влево или вправо, чтобы просмотреть другие элементы списка. Обратите внимание: переключение страниц происходит плавно и без задержек. По умолчанию ViewPager загружает элемент, находящийся на экране, а также по одному соседнему элементу в каждом направлении, чтобы отклик на жест прокрутки был немедленным. Количество загружаемых соседних страниц можно настроить вызовом setOffscreenPageLimit(int).

Однако с ViewPager еще не все идеально. Вернитесь к списку при помощи кнопки Back и нажмите на другом элементе. Вы снова увидите информацию первого элемента — вместо того, который был запрошен.

По умолчанию ViewPager отображает в своем экземпляре PagerAdapter первый элемент. Чтобы вместо него отображался элемент, выбранный пользователем, назначьте текущим элементом ViewPager элемент с указанным индексом.

В конце CrimePagerActivity.onCreate(…) найдите индекс отображаемого преступления; для этого переберите и проверьте идентификаторы всех преступлений. Когда вы найдете экземпляр Crime, у которого поле mId совпадает с crimeId в дополнении интента, измените текущий элемент по индексу найденного объекта Crime.

Листинг 11.7. Назначение исходного элемента (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        FragmentManager fm = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
            ...
        });
```

продолжение ⇏

Листинг 11.7 (*продолжение*)

```
        UUID crimeId = (UUID)getIntent()
            .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);
        for (int i = 0; i < mCrimes.size(); i++) {
            if (mCrimes.get(i).getId().equals(crimeId)) {
                mViewPager.setCurrentItem(i);
                break;
            }
        }
    }
}
```

Запустите приложение CriminalIntent. При выборе любого элемента списка должна отображаться подробная информация правильного объекта Crime.

Также в приложение можно добавить еще одну функцию: заменить заголовок активности, выводимый на панели действий (строки заголовка на старых устройствах), кратким описанием текущего объекта Crime. Для этого следует реализовать интерфейс ViewPager.OnPageChangeListener.

Листинг 11.8. Добавление OnPageListener (CrimePagerActivity.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mViewPager = new ViewPager(this);
    mViewPager.setId(R.id.viewPager);
    setContentView(mViewPager);

    mCrimes = CrimeLab.get(this).getCrimes();

    FragmentManager fm = getSupportFragmentManager();
    mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
        ...
    });

    mViewPager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {
        public void onPageScrollStateChanged(int state) { }
        public void onPageScrolled(int pos, float posOffset, int posOffsetPixels) {
}
        public void onPageSelected(int pos) {
            Crime crime = mCrimes.get(pos);
            if (crime.getTitle() != null) {
                setTitle(crime.getTitle());
            }
        }
    });
    ...
}
```

Метод onPageChangeListener используется для обнаружения изменений в странице, которая в настоящий момент отображается экземпляром ViewPager. При изменении страницы заголовку CrimePagerActivity задается краткое описание Crime.

Нас интересует лишь то, какая страница является текущей, поэтому мы реализуем метод onPageSelected(…). Метод onPageScrolled(…) сообщает, где будет находиться

страница, а метод `onPageScrollStateChanged(…)` — находится ли анимация страницы в процессе активного перетаскивания, перехода в устойчивое состояние или в простое.

Запустите приложение CriminalIntent и убедитесь в том, что при каждом жесте прокрутки заголовок активности заполняется значением поля `mTitle` текущего объекта `Crime`. Вот и все! Теперь наш экземпляр `ViewPager` полностью готов к работе.

FragmentStatePagerAdapter и FragmentPagerAdapter

Существует еще один тип `PagerAdapter`, который можно использовать в приложениях; он называется `FragmentPagerAdapter`.

`FragmentPagerAdapter` используется точно так же, как `FragmentStatePagerAdapter`, и отличается от него только способом выгрузки неиспользуемых фрагментов.

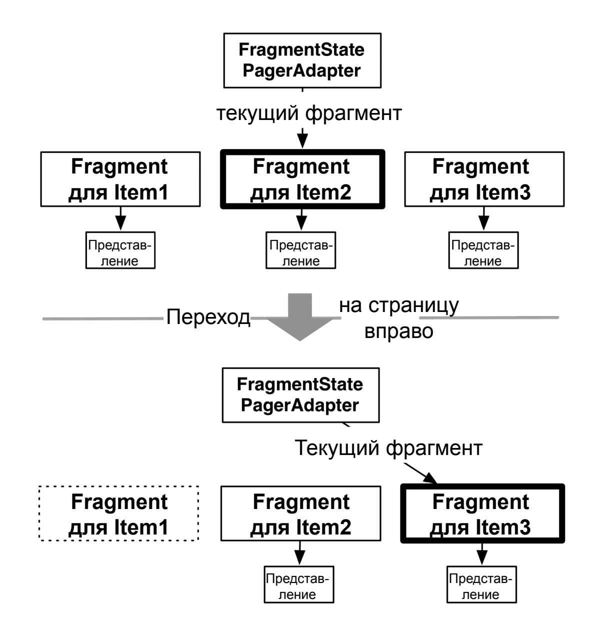


Рис. 11.3. Управление фрагментами FragmentStatePagerAdapter

При использовании класса `FragmentStatePagerAdapter` неиспользуемый фрагмент уничтожается. Происходит закрепление транзакции для полного удаления фрагмента из объекта `FragmentManager` активности. Наличие «состояния» у `FragmentStatePagerAdapter` определяется тем фактом, что экземпляр при уничтожении сохраняет объект `Bundle` вашего фрагмента в методе `onSaveInstanceState(Bundle)`. Когда пользователь возвращается обратно, новый фрагмент восстанавливается по состоянию этого экземпляра.

С другой стороны, `FragmentPagerAdapter` ничего подобного не делает. Когда фрагмент становится ненужным, `FragmentPagerAdapter` вызывает для транзакции `detach(Fragment)` вместо `remove(Fragment)`. Представление фрагмента при этом

уничтожается, но экземпляр фрагмента продолжает существовать в `FragmentManager`. Таким образом, фрагменты, созданные `FragmentPagerAdapter`, никогда не уничтожаются.

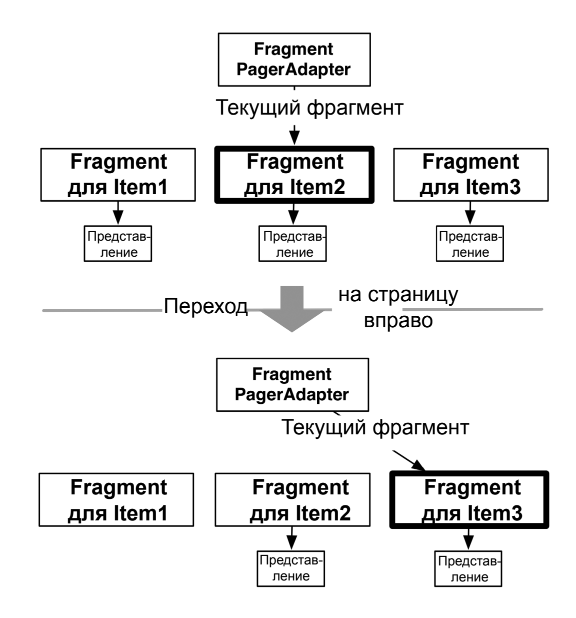


Рис. 11.4. Управление фрагментами FragmentPagerAdapter

Выбор используемого адаптера зависит от приложения. Как правило, `FragmentStatePagerAdapter` более экономно расходует память. Приложение CriminalIntent выводит список, который со временем может стать достаточно длинным, причем к каждому элементу списка может прилагаться фотография. Хранить всю эту информацию в памяти нежелательно, поэтому мы используем`FragmentStatePagerAdapter`.

С другой стороны, если интерфейс содержит небольшое фиксированное количество фрагментов, использование `FragmentPagerAdapter` безопасно и уместно. Самый характерный пример такого рода — интерфейс с вкладками. Некоторые детализированные представления не помещаются на одном экране, поэтому отображаемая информация распределяется между несколькими вкладками. Добавление `ViewPager` с перебором вкладок делает этот интерфейс интуитивным. Хранение фрагментов в памяти упрощает управление кодом контроллера, а поскольку этот стиль интерфейса обычно использует всего два или три фрагмента на активность, проблемы с нехваткой памяти крайне маловероятны.

Для любознательных: как работает ViewPager

Классы `ViewPager` и `PagerAdapter` незаметно выполняют большую часть рутинной работы. В этом разделе приведена более подробная информация о том, что при этом происходит.

Пара предупреждений, прежде чем мы перейдем к обсуждению: во-первых, согласно документации класс ViewPager все еще находится в процессе разработки, так что интерфейсы могут измениться в будущем. Во-вторых, в большинстве случаев понимать все технические подробности не обязательно.

Но если вы захотите реализовать интерфейс PagerAdapter самостоятельно, вы должны знать, чем отношения ViewPager-PagerAdapter отличаются от обычных отношений AdapterView-Adapter.

Почему ViewPager, а не AdapterView? У AdapterView имеется субкласс Gallery с похожим поведением. Почему бы не использовать его?

Использование AdapterView в данном случае потребует большого объема работы, потому что вы не сможете использовать существующие экземпляры Fragment. Adapter ожидает, что вы сможете предоставить View мгновенно. Но когда будет создано представление вашего фрагмента, решает FragmentManager, а не вы. Таким образом, когда Gallery обратиться к Adapter за представлением вашего фрагмента, вы не сможете создать фрагмент *и* немедленно выдать его представление.

Именно по этой причине и существует класс ViewPager. Вместо Adapter он использует класс с именем PagerAdapter. Этот класс сложнее Adapter, потому что он выполняет больший объем работы по управлению представлениями. Ниже кратко перечислены основные различия.

Вместо метода getView(…), возвращающего представление, PagerAdapter содержит следующие методы:

```
public Object instantiateItem(ViewGroup container, int position)
public void destroyItem(ViewGroup container, int position, Object object)
public abstract boolean isViewFromObject(View view, Object object)
```

Метод pagerAdapter.instantiateItem(ViewGroup, int) приказывает адаптеру создать представление элемента списка для заданной позиции и добавить его в контейнер ViewGroup; метод destroyItem(ViewGroup, int, Object) приказывает уничтожить этот элемент. Обратите внимание: метод instantiateItem(ViewGroup, int) не приказывает создать представление немедленно. PagerAdapter может создать представление в будущем.

После того как представление было создано, ViewPager в какой-то момент замечает его. Чтобы понять, к какому элементу списка оно относится, ViewPager вызывает метод isViewFromObject(View, Object). Параметр Object содержит объект, полученный при вызове instantiateItem(ViewGroup, int). Таким образом, если ViewPager вызывает instantiateItem(ViewGroup, 5) и получает объект A, вызов isViewFromObject(View, A) должен вернуть true, если переданный экземпляр View относится к элементу 5, и false в противном случае.

Этот процесс достаточно сложен для ViewPager, но не для класса PagerAdapter, который должен уметь только создавать представления, уничтожать представления и определять, к какому объекту относится представление. Менее жесткие требования позволяют реализации PagerAdapter создавать и добавлять новый фрагмент в instantiateItem(ViewGroup, int) и возвращать фрагмент как отслеживаемый экземпляр Object. При этом isViewFromObject(View, Object) выглядит примерно так:

```
@Override
public boolean isViewFromObject(View view, Object object) {
    return ((Fragment)object).getView() == view;
}
```

Реализовать переопределения `PagerAdapter` каждый раз, когда потребуется использовать `ViewPager`, было бы слишком утомительно. Хорошо, что у нас есть `FragmentPagerAdapter` и `FragmentStatePagerAdapter`.

12 Диалоговые окна

Диалоговые окна требуют внимания пользователя и ввода данных. Обычно они используются для принятия решений или отображения важной информации. В этой главе мы добавим диалоговое окно, в котором пользователь может изменить дату преступления.

При нажатии кнопки даты в `CrimeFragment` открывается диалоговое окно, показанное на рис. 12.1.

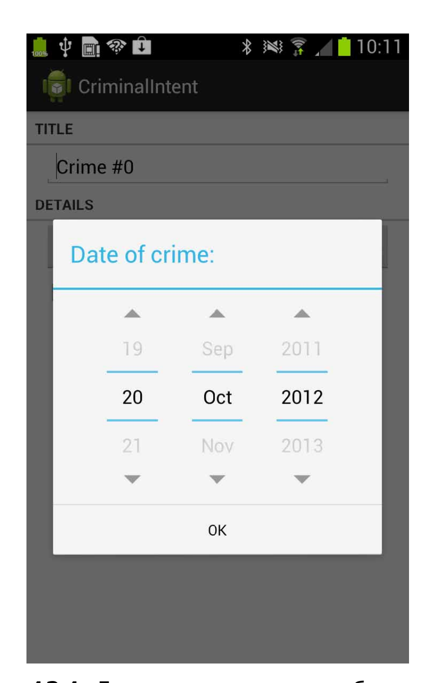


Рис. 12.1. Диалоговое окно для выбора даты

Диалоговое окно на рис. 12.1 является экземпляром AlertDialog — субкласса Dialog. Именно этот многоцелевой субкласс Dialog вы будете чаще всего использовать в своих программах.

(У AlertDialog существует субкласс DatePickerDialog, который вроде бы идеально подходит для наших целей. Однако на момент написания книги реализация DatePickerDialog работала с ошибками; использовать AlertDialog проще, чем обходить эти ошибки.)

Экземпляр AlertDialog на рис. 12.1 упакован в экземпляр DialogFragment — субкласса Fragment. Вообще говоря, экземпляр AlertDialog может отображаться и без DialogFragment, но Android так поступать не рекомендует. Управление диалоговым окном из FragmentManager открывает больше возможностей для его отображения.

Кроме того, «минимальный» экземпляр AlertDialog исчезнет при повороте устройства. С другой стороны, если экземпляр AlertDialog упакован в фрагмент, после поворота диалоговое окно будет создано заново и появится на экране.

Для приложения CriminalIntent мы создадим субкласс DialogFragment с именем DatePickerFragment. В коде DatePickerFragment создается и настраивается экземпляр AlertDialog, отображающий виджет DatePicker. В качестве хоста DatePickerFragment используется экземпляр CrimePagerActivity.

На рис. 12.2 изображена схема этих отношений.

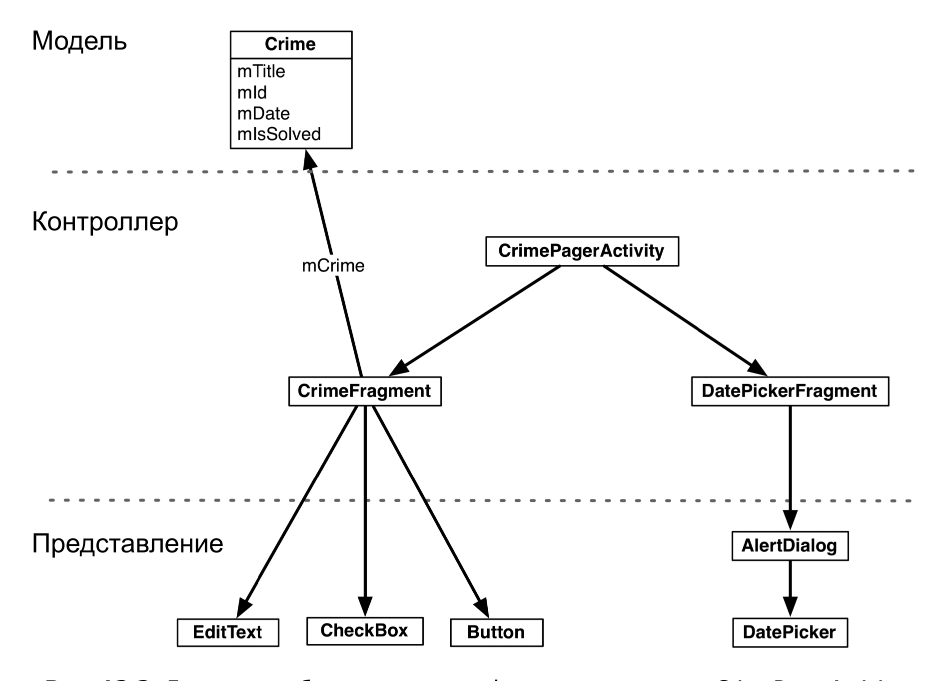


Рис. 12.2. Диаграмма объектов для двух фрагментов с хостом CrimePagerActivity

Наши первоочередные задачи:

- создание класса `DatePickerFragment`;
- построение `AlertDialog`;
- вывод диалогового окна на экран с использованием `FragmentManager`.

Позднее в этой главе мы подключим виджет `DatePicker` и организуем передачу необходимых данных между `CrimeFragment` и `DatePickerFragment`.

Прежде чем браться за работу, добавьте строковый ресурс (листинг 12.1).

Листинг 12.1. Добавление строки для заголовка диалогового окна (values/strings.xml)

```
<resources>

  ...
  <string name="crime_solved_label">Solved?</string>
  <string name="crimes_title">Crimes</string>
  <string name="date_picker_title">Date of crime:</string>

</resources>
```

Создание DialogFragment

Создайте новый класс с именем `DatePickerFragment` и назначьте его суперклассом версию `DialogFragment` из библиотеки поддержки: `android.support.v4.app.DialogFragment`.

Класс `DialogFragment` содержит следующий метод:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

Экземпляр `FragmentManager` активности-хоста вызывает этот метод в процессе вывода `DialogFragment` на экран.

Добавьте в файл DatePickerFragment.java реализацию `onCreateDialog(…)`, которая создает `AlertDialog` с заголовком и одной кнопкой OK. (Виджет `DatePicker` мы добавим позднее.)

Листинг 12.2. Создание DialogFragment (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

В этой реализации используется класс `AlertDialog.Builder`, предоставляющий динамичный интерфейс для конструирования экземпляров `AlertDialog`.

Сначала мы передаем объект `Context` конструктору `AlertDialog.Builder`, который возвращает экземпляр `AlertDialog.Builder`.

Затем вызываются два метода AlertDialog.Builder для настройки диалогового окна:

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

Метод setPositiveButton(…) получает строковый ресурс и объект, реализующий DialogInterface.OnClickListener. В листинге 12.2 передается константа Android для кнопки OK и null вместо слушателя. Слушатель будет реализован позднее в этой главе.

Положительная кнопка (Positive) нажимается пользователем для подтверждения информации в диалоговом окне. В AlertDialog также можно добавить еще две кнопки: отрицательную (Negative) и нейтральную (Neutral). Эти обозначения определяют позицию кнопок в диалоговом окне (если их несколько). На устройствах Froyo и Gingerbread положительной является левая кнопка. На более новых устройствах порядок кнопок изменен, и положительной является правая кнопка).

Построение диалогового окна завершается вызовом AlertDialog.Builder.create(), который возвращает настроенный экземпляр AlertDialog.

На этом возможности AlertDialog и AlertDialog.Builder не исчерпаны; подробности достаточно хорошо изложены в документации разработчика. А пока давайте перейдем к механике вывода диалогового окна на экран.

Отображение DialogFragment

Как и все фрагменты, экземпляры DialogFragment находятся под управлением экземпляра FragmentManager активности-хоста.

Для добавления экземпляра DialogFragment в FragmentManager и вывода его на экран используются следующие методы экземпляра фрагмента:

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

Строковый параметр однозначно идентифицирует DialogFragment в списка FragmentManager. Выбор версии (с FragmentManager или FragmentTransaction) зависит только от вас — если передать FragmentManager, транзакция будет автоматически создана и закреплена. В нашем примере передается FragmentManager.

Добавьте в CrimeFragment константу для метки DatePickerFragment. Затем в методе onCreateView(…) удалите код, блокирующий кнопку даты, и назначьте слушателя View.OnClickListener, который отображает DatePickerFragment при нажатии кнопки даты.

Листинг 12.3. Отображение DialogFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";
    ...
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                        .getSupportFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(fm, DIALOG_DATE);
            }
        });
        mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
        ...

        return v;
    }

    ...
}
```

Запустите приложение CriminalIntent и нажмите кнопку даты, чтобы диалоговое окно появилось на экране. Кнопка OK закрывает диалоговое окно (рис. 12.3).

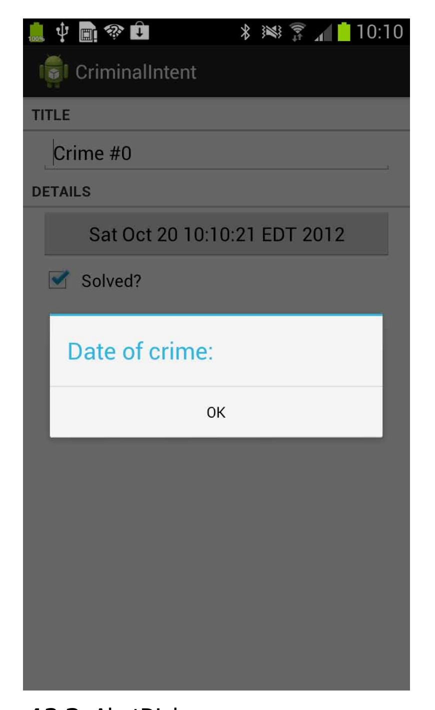


Рис. 12.3. AlertDialog с заголовком и кнопкой

Назначение содержимого диалогового окна

Далее мы включим в AlertDialog виджет DatePicker при помощи метода AlertDialog.Builder:

```
public AlertDialog.Builder setView(View view)
```

Метод настраивает диалоговое окно для отображения переданного объекта View между заголовком и кнопкой(-ами).

В Package Explorer создайте новый файл макета с именем dialog_date.xml и назначьте его корневым элементом DatePicker. Макет будет состоять из одного объекта View (DatePicker), который мы заполним и передадим setView(…).

Настройте макет DatePicker так, как показано на рис. 12.4.

DatePicker
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:id="**@+id/dialog_date_datePicker**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:calendarViewShown="**false**"

Рис. 12.4. Макет DatePicker (layout/dialog_date.xml)

В методе DatePickerFragment.onCreateDialog(…) заполните представление и назначьте его диалоговому окну.

Листинг 12.4. Включение DatePicker в AlertDialog (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

Запустите приложение CriminalIntent. Нажмите кнопку даты и убедитесь в том, что в диалоговом окне теперь отображается DatePicker.

Почему мы возимся с определением и заполнением макета, когда объект DatePicker можно было бы создать в коде так, как показано ниже?

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker dp = new  DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(dp)
        ...
        .create();
}
```

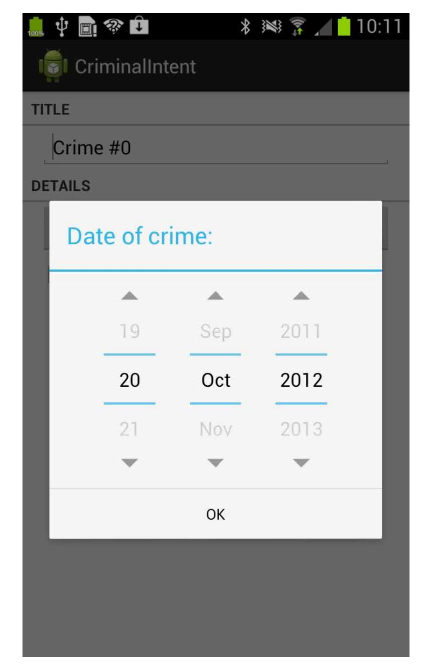


Рис. 12.5. AlertDialog с DatePicker

Использование макета упрощает изменения в случае изменения его содержимого. Предположим, вы захотели, чтобы рядом с `DatePicker` в диалоговом окне отображался виджет `TimePicker`. При использовании заполнения можно просто обновить файл макета, и новое представление появится на экране.

Итак, наше диалоговое окно успешно отображается. В следующем разделе мы свяжем его с полем даты `Crime` и позаботимся о том, чтобы пользователь мог вводить данные.

Передача данных между фрагментами

Мы передавали данные между двумя активностями; мы передавали данные между двумя фрагментными активностями. Теперь нужно передать данные между двумя фрагментами, хостом которых является одна активность — `CrimeFragment` и `DatePickerFragment` (см. рис. 5.10).

Чтобы передать дату преступления `DatePickerFragment`, мы напишем метод `newInstance(Date)` и сделаем объект `Date` аргументом фрагмента.

Чтобы вернуть новую дату фрагменту `CrimeFragment` для обновления уровня модели и его собственного представления, мы упакуем ее как дополнение объекта `Intent` и передадим этот объект `Intent` в вызове `CrimeFragment.onActivityResult(…)`.

Вызов `Fragment.onActivityResult(…)` может показаться странным — с учетом того, что активность-хост не получает вызова `Activity.onActivityResult(…)` в этом взаимодействии. Но как будет показано позднее в этой главе, использование `onActivityResult(…)` для передачи данных от одного фрагмента к другому не только работает, но и улучшает гибкость отображения фрагмента диалогового окна.

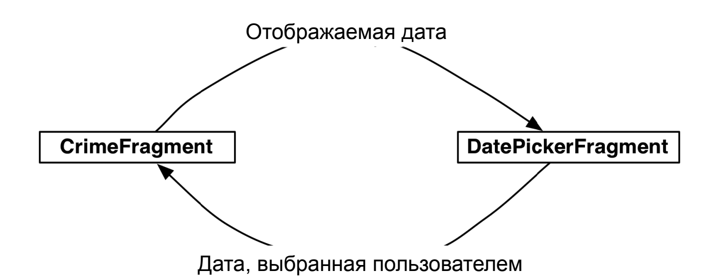


Рис. 12.6. Взаимодействие между CrimeFragment и DatePickerFragment

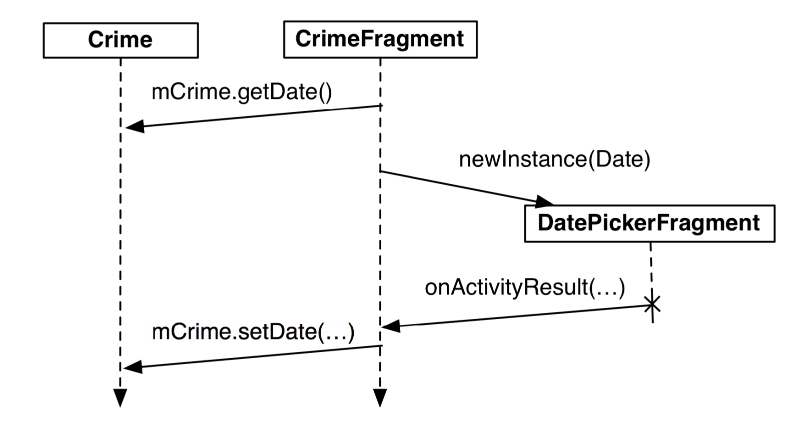


Рис. 12.7. Последовательность событий взаимодействия между CrimeFragment и DatePickerFragment

Передача данных DatePickerFragment

Чтобы получить данные в `DatePickerFragment`, мы сохраним дату в пакете аргументов `DatePickerFragment`, где `DatePickerFragment` сможет обратиться к ней.

Создание аргументов фрагмента и присваивание им значений обычно выполняется в методе `newInstance()`, заменяющем конструктор фрагмента. Добавьте в файл DatePickerFragment.java метод `newInstance(Date)`.

Листинг 12.5. Добавление метода newInstance(Date) (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    public static final String EXTRA_DATE =
        "com.bignerdranch.android.criminalintent.date";

    private Date mDate;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_DATE, date);
```

```
        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);

        return fragment;
    }

    ...
}
```

В классе CrimeFragment удалите вызов конструктора DatePickerFragment и замените его вызовом DatePickerFragment.newInstance(Date).

Листинг 12.6. Добавление вызова newInstance() (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater,
        ViewGroup parent, Bundle savedInstanceState) {
    ...

    mDateButton = (Button)v.findViewById(R.id.crime_date);
    updateDate();
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(fm, DIALOG_DATE);
        }
    });

    return v;
}
```

Экземпляр DatePickerFragment должен инициализировать DatePicker по информации, хранящейся в Date. Однако для инициализации DatePicker необходимо иметь целочисленные значения месяца, дня и года. Объект Date больше напоминает временную метку и не может предоставить нужные целые значения напрямую.

Чтобы получить нужные значения, следует создать объект Calendar и использовать Date для определения его конфигурации. После этого вы сможете получить нужную информацию из Calendar.

В методе onCreateDialog(…) получите объект Date из аргументов и используйте его с Calendar для инициализации DatePicker.

Листинг 12.7. Извлечение Date и инициализация DatePicker (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mDate = (Date)getArguments().getSerializable(EXTRA_DATE);

    // создание объекта Calendar для получения года, месяца и дня
    Calendar calendar = Calendar.getInstance();
```

продолжение ⇗

Листинг 12.7 (*продолжение*)

```
    calendar.setTime(mDate);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    DatePicker datePicker = (DatePicker)v.findViewById(R.id.dialog_date_datePicker);
    datePicker.init(year, month, day, new OnDateChangedListener() {
        public void onDateChanged(DatePicker view, int year, int month, int day) {
            // Преобразование года, месяца и дня в объект Date
            mDate = new GregorianCalendar(year, month, day).getTime();

            // обновление аргумента для сохранения
            // выбранного значения при повороте
            getArguments().putSerializable(EXTRA_DATE, mDate);
        }
    });
    ...
}
```

При инициализации DatePicker мы также назначаем ему слушателя OnDateChangedListener. Когда пользователь изменяет дату в DatePicker, мы обновляем Date в соответствии с внесенными изменениями. В следующем разделе объект Date будет возвращаться CrimeFragment.

В конце onDateChanged(…) объект Date записывается обратно в аргументы фрагмента. Это делается для того, чтобы сохранить значение mDate в случае поворота. Если в момент поворота устройства DatePickerFragment находится на экране, FragmentManager уничтожает текущий экземпляр и создает новый. При создании нового экземпляра FragmentManager вызывает для него onCreateDialog(…), и экземпляр получает сохраненную дату из аргументов. Это более простой способ сохранения значения, чем сохранение состояния в onSaveInstanceState(…).

(У читателей, имеющих больше опыта работы с фрагментами, может возникнуть вопрос: почему бы не сохранить DatePickerFragment? Сохранение фрагментов — отличный способ обработки поворотов, который будет более подробно рассматриваться в главе 14. К сожалению, в настоящее время реализация DialogFragment содержит ошибку, из-за которой сохраненные экземпляры работают некорректно, так что сохранение DatePickerFragment нельзя признать хорошим решением.)

Сейчас CrimeFragment успешно сообщает DatePickerFragment, какую дату следует отобразить. Вы можете запустить приложение CriminalIntent и убедиться в том, что все работает так же, как прежде.

Возвращение данных CrimeFragment

Чтобы экземпляр CrimeFragment получал данные от DatePickerFragment, нам необходимо каким-то образом отслеживать отношения между двумя фрагментами.

С активностями вы вызываете `startActivityForResult(…)`, а `ActivityManager` отслеживает отношения между родительской и дочерней активностью. Когда дочерняя активность прекращает существование, `ActivityManager` знает, какая активность должна получить результат.

Назначение целевого фрагмента

Для создания аналогичной связи можно назначить `CrimeFragment` *целевым фрагментом* (target fragment) для `DatePickerFragment`. Для этого вызывается следующий метод `Fragment`:

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

Метод получает фрагмент, который станет целевым, и код запроса, аналогичный передаваемому `startActivityForResult(…)`. По коду запроса целевой фрагмент позднее может определить, какой фрагмент возвращает информацию.

`FragmentManager` сохраняет целевой фрагмент и код запроса. Чтобы получить их, вызовите `getTargetFragment()` и `getTargetRequestCode()` для фрагмента, назначившего целевой фрагмент.

В файле CrimeFragment.java создайте константу для кода запроса, а затем назначьте `CrimeFragment` целевым фрагментом экземпляра `DatePickerFragment`.

Листинг 12.8. Назначение целевого фрагмента (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";
    private static final int REQUEST_DATE = 0;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                        .getSupportFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                    .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(fm, DIALOG_DATE);
            }
        });

        return v;
    }

    ...
}
```

Передача данных целевому фрагменту

Итак, связь между CrimeFragment и DatePickerFragment создана, и теперь нужно вернуть дату CrimeFragment. Дата будет включена в объект Intent как дополнение.

Какой метод будет использоваться для передачи интента целевому фрагменту? Как ни странно, DatePickerFragment передаст его при вызове CrimeFragment.onActivityResult(int, int, Intent).

Метод Activity.onActivityResult(…) вызывается ActivityManager для родительской активности при уничтожении дочерней активности. При работе с активностями вы не вызываете Activity.onActivityResult(…) самостоятельно; это делает ActivityManager. После того как активность получит вызов, экземпляр FragmentManager активности вызывает Fragment.onActivityResult(…) для соответствующего фрагмента.

Если хостом двух фрагментов является одна активность, то для возвращения данных можно воспользоваться методом Fragment.onActivityResult(…) и вызывать его непосредственно для целевого фрагмента. Он содержит все необходимое:

- код запроса, соответствующий коду, переданному setTargetFragment(…), по которому целевой фрагмент узнает, кто возвращает результат;
- код результата для определения выполняемого действия;
- экземпляр Intent, который может содержать дополнительные данные.

В классе DatePickerFragment создайте закрытый метод, который создает интент, помещает в него дату как дополнение, а затем вызывает CrimeFragment.onActivityResult(…). В onCreateDialog(…) замените параметр null вызова setPositiveButton(…) реализацией DialogInterface.OnClickListener, которая вызывает закрытый метод и передает код результата.

Листинг 12.9. Обратный вызов целевого фрагмента (DatePickerFragment.java)

```
private void sendResult(int resultCode) {
    if (getTargetFragment() == null)
        return;

    Intent i = new Intent();
    i.putExtra(EXTRA_DATE, mDate);

    getTargetFragment()
        .onActivityResult(getTargetRequestCode(), resultCode, i);
}

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .setPositiveButton(
            android.R.string.ok,
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
```

```
                    sendResult(Activity.RESULT_OK);
                }
            })
        .create();
}
```

В классе `CrimeFragment` переопределите метод `onActivityResult(…)`, чтобы он возвращал дополнение, задавал дату в `Crime` и обновлял текст кнопки даты.

Листинг 12.10. Реакция на получение данных от диалогового окна (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        Date date = (Date)data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}
```

Код, задающий текст кнопки, идентичен коду из `onCreateView(…)`. Чтобы избежать задания текста в двух местах, мы инкапсулируем этот код в закрытом методе `updateDate()`, а затем вызовем его в `onCreateView(…)` и `onActivityResult(…)`.

Листинг 12.11. Выделение кода в метод updateDate() (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
   ...

    public void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        ...
        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        updateDate();

    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;
        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
            updateDate();
        }
    }
}
```

Круг замкнулся — данные передаются туда и обратно.

Запустите приложение CriminalIntent и убедитесь в том, что вы действительно можете управлять датой. Измените дату `Crime`; новая дата должна появиться в представлении `CrimeFragment`. Вернитесь к списку преступлений и проверьте дату `Crime` и убедитесь в том, что уровень модели действительно обновлен.

Больше гибкости в представлении DialogFragment

Использование `onActivityResult(…)` для возвращения данных целевому фрагменту особенно удобно, когда ваше приложение получает много данных от пользователя и нуждается в большем пространстве для их ввода. При этом приложение должно хорошо работать на телефонах и планшетах.

На экране телефона свободного места не так много, поэтому вы, скорее всего, используете для ввода данных активность с полноэкранным фрагментом. Дочерняя активность будет запускаться вызовом `startActivityForResult()` из фрагмента родительской активности. При уничтожении дочерней активности родительская активность будет получать вызов `onActivityResult(…)`, который будет перенаправляться фрагменту, запустившему дочернюю активность.

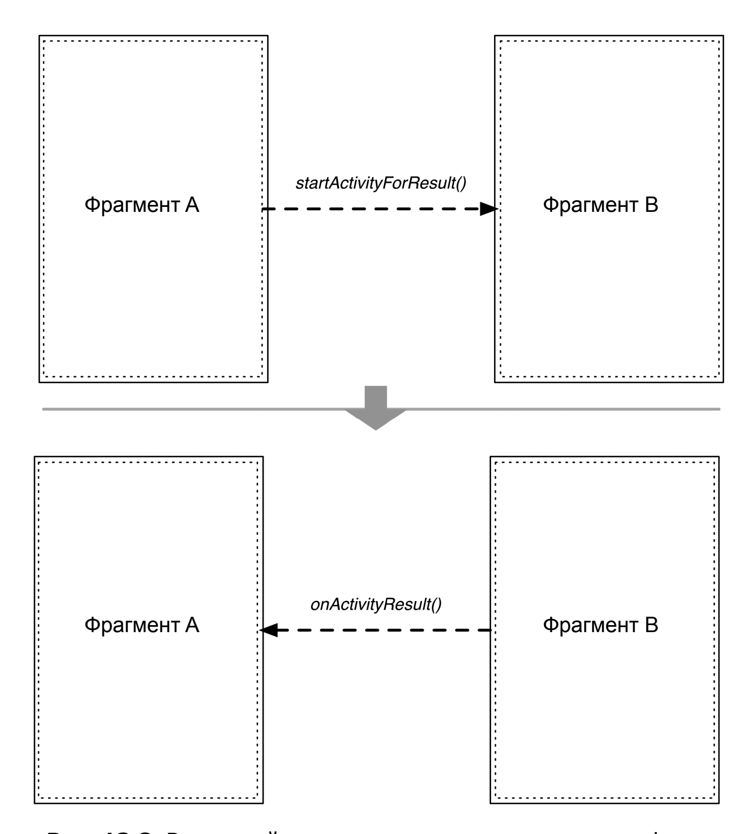


Рис. 12.8. Взаимодействие между активностями на телефонах

На планшетах, где экранного пространства больше, часто бывает лучше отобразить `DialogFragment` для ввода тех же данных. В таком случае вы задаете целевой фрагмент и вызываете `show(…)` для фрагмента диалогового окна. При закрытии фрагмент диалогового окна вызывает для своей цели `onActivityResult(…)`.

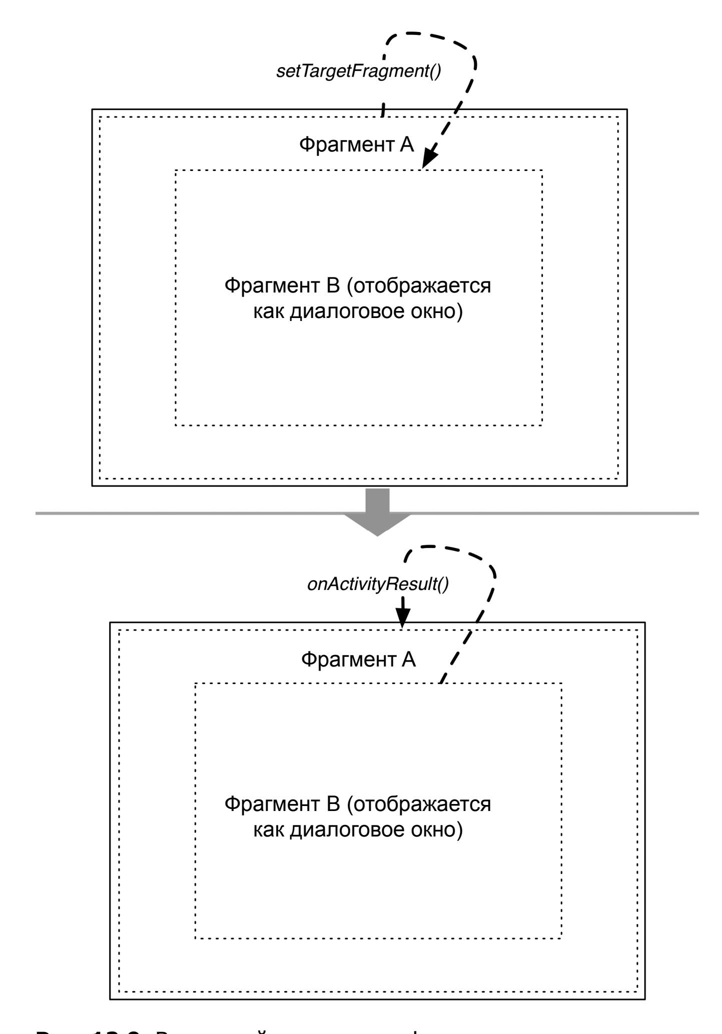


Рис. 12.9. Взаимодействие между фрагментами на планшетах

Метод `onActivityResult(…)` фрагмента будет вызываться всегда, независимо от того, запустил ли фрагмент активность или отобразил диалоговое окно. Следовательно, мы можем использовать один код для разных вариантов представления информации.

Когда один код используется и для полноэкранного, и для диалогового фрагмента, для подготовки вывода в обоих случаях вместо `onCreateDialog(…)` можно переопределить `DialogFragment.onCreateView(…)`.

Упражнение. Новые диалоговые окна

В качестве несложного упражнения напишите еще один диалоговый фрагмент `TimePickerFragment` для выбора времени преступления. Используйте виджет `TimePicker`, добавьте в `CrimeFragment` еще одну кнопку для отображения `TimePickerFragment`.

Если вам захочется усложнить задачу, попробуйте ограничиться однокнопочным интерфейсом. Диалоговое окно, открываемое кнопкой, должно предлагать пользователю выбрать между изменением времени и изменением даты. После выбора на экране должно появляться второе диалоговое окно.

13 Воспроизведение звука и MediaPlayer

В следующих трех главах мы оставим CriminalIntent в покое и построим другое приложение. Оно будет воспроизводить исторические аудиофайлы с использованием класса MediaPlayer.

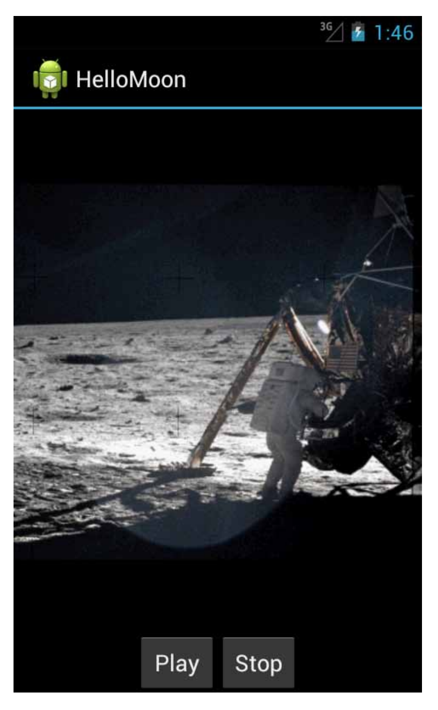


Рис. 13.1. Привет, Луна!

`MediaPlayer` — класс Android для воспроизведения аудио- и видеоданных. Он может воспроизводить данные из разных источников (например, локальных файлов или файлов, загружаемых из Интернета) и в разных форматах (WAV, MP3, Ogg Vorbis, MPEG-4, 3GPP и т. д.).

Создайте новый проект с именем HelloMoon. В первом диалоговом окно выберите тему приложения Holo Dark.

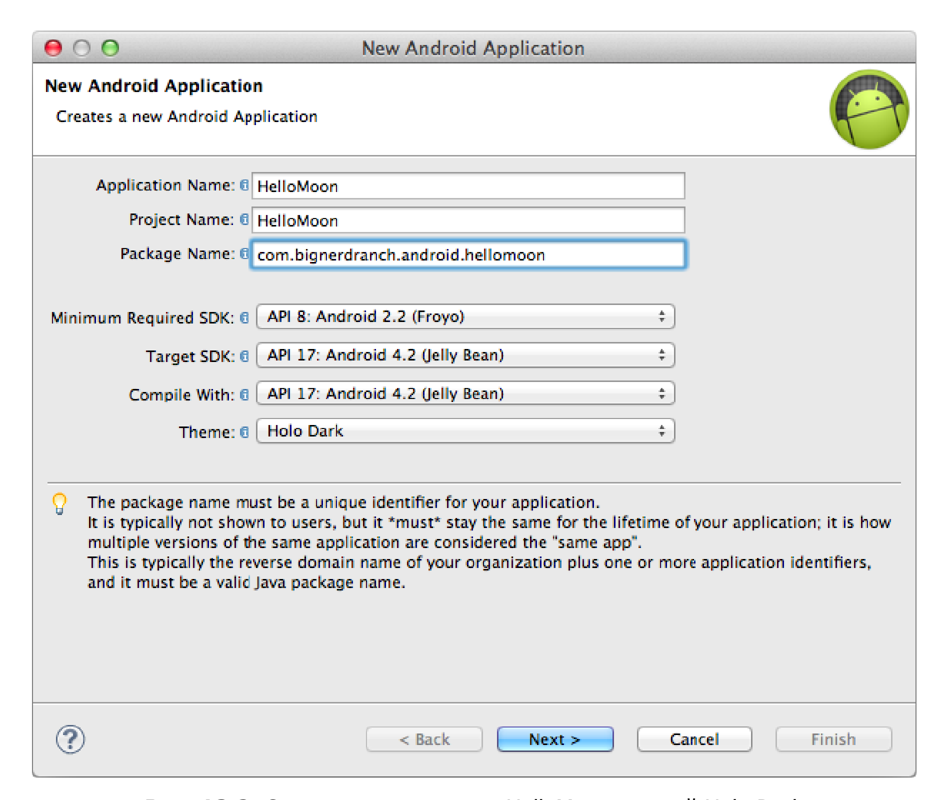


Рис. 13.2. Создание приложения HelloMoon с темой Holo Dark

Щелкните на кнопке Next. У приложения не будет собственного значка лаунчера, и оно использует пустой шаблон активности. Прикажите шаблону создать активность с именем `HelloMoonActivity`.

Добавление ресурсов

Для приложения HelloMoon необходимы файлы (графический и звуковой) из архива решений этой книги (http://www.bignerdranch.com/solutions/AndroidProgramming.zip). Найдите в архиве решений следующие файлы:

- 13_Audio/HelloMoon/res/drawable-mdpi/armstrong_on_moon.jpg
- 13_Audio/HelloMoon/res/raw/one_small_step.wav

Для этого простого приложения мы создали один файл armstrong_on_moon.jpg для экранов средней плотности (~160 dpi), которую Android считает минимальным общим требованием. Скопируйте файл armstrong_on_moon.jpg в каталог drawable-mdpi.

Аудиофайл будет находиться в каталоге res/raw. Каталог raw предназначен для хранения любых ресурсов, которые не требуют специальной обработки системой построения приложений Android.

Каталог res/raw не создается для проекта по умолчанию, поэтому его придется добавить вручную. (Щелкните правой кнопкой мыши на каталоге res и выберите команду New ▸ Folder.) Скопируйте файл one_small_step.wav в новый каталог.

(Заодно скопируйте в каталог res/raw/ файл 13_Audio/HelloMoon/res/raw/apollo_17_stroll.mpg. Этот файл будет использоваться для воспроизведения видео в упражнении в конце этой главы.)

Откройте файл res/values/strings.xml и добавьте строки, необходимые для приложения HelloMoon:

Листинг 13.1. Добавление строк (strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">HelloMoon</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="hellomoon_play">Play</string>
  <string name="hellomoon_stop">Stop</string>
  <string name="hellomoon_description">Neil Armstrong stepping
          onto the moon</string>

</resources>
```

(Почему в приложении HelloMoon вместо значков на кнопках используются строки «Play» и «Stop»? В главе 15 мы займемся локализацией этого приложения, а со значками это будет не так интересно.)

Итак, необходимые ресурсы готовы; можно переходить к планированию общей архитектуры HelloMoon.

В приложении HelloMoon будет использоваться только одна активность `HelloMoonActivity`, являющаяся хостом для фрагмента `HelloMoonFragment`.

`AudioPlayer` — класс, который мы напишем для инкапсуляции `MediaPlayer`. Вообще говоря, инкапсулировать `MediaPlayer` не обязательно; `HelloMoonFragment` может взаимодействие с `MediaPlayer` напрямую. Тем не менее такая архитектура делает код более стройным и ослабляет логические привязки.

Но прежде чем создавать класс `AudioPlayer`, мы сначала подготовим остальные части приложения. К этому моменту вы уже должны достаточно хорошо представлять себе следующие фазы:

- определение макета фрагмента;
- создание класса фрагмента;
- изменение активности и ее макета для выполнения функций хоста фрагмента.

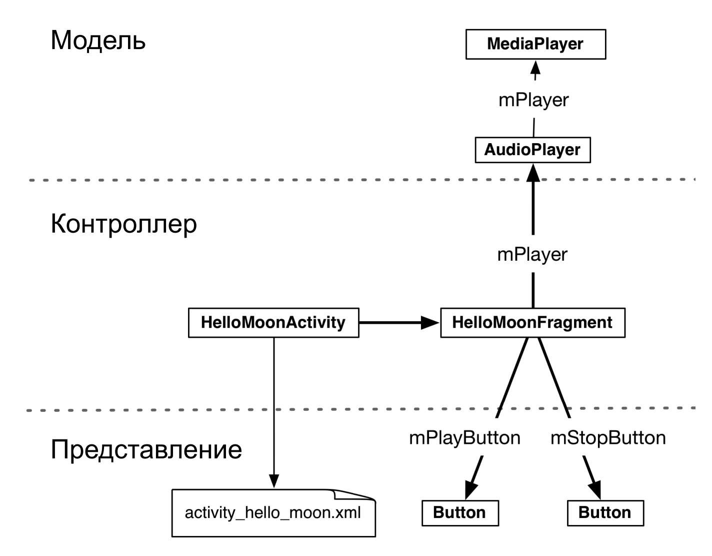


Рис. 13.3. Диаграмма объектов HelloMoon

Определение макета HelloMoonFragment

Создайте новый XML-файл макета Android с именем fragment_hello_moon.xml. Назначьте его корневым элементом `TableLayout`.

Файл fragment_hello_moon.xml следует заполнять по рис. 13.4.

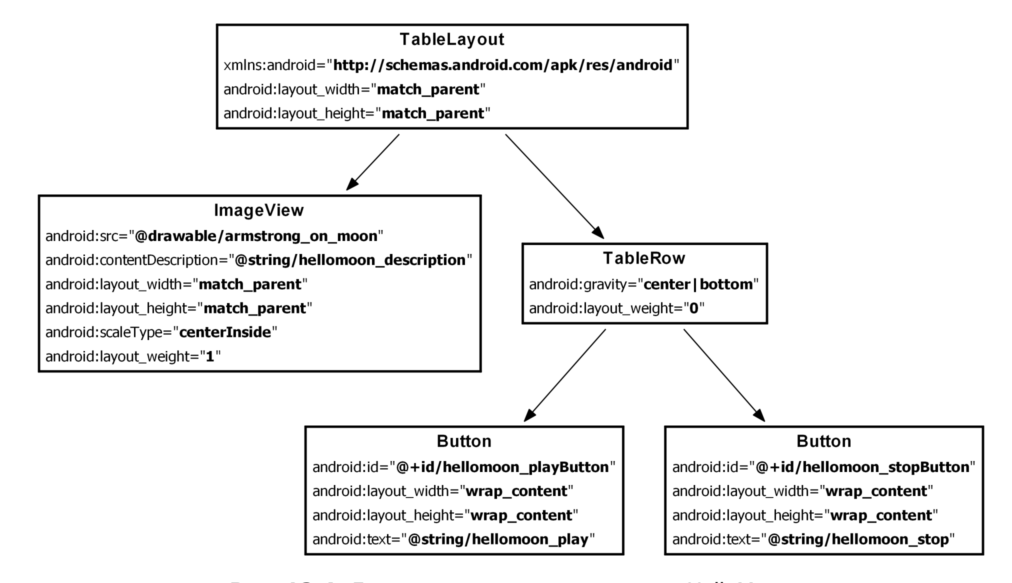


Рис. 13.4. Диаграмма макета приложения HelloMoon

TableLayout работает почти так же, как LinearLayout. Вместо вложения LinearLayout для упорядочения виджетов можно воспользоваться виджетом TableRow. Комбинация TableLayout и TableRow упрощает организацию представлений в аккуратные столбцы.

Почему ImageView не находится в TableRow? Потомки TableRow рассматриваются как «ячейки» таблицы. Мы хотим, чтобы виджет ImageView занимал весь экран. Если бы он был потомком TableRow, то виджет TableLayout постарается развернуть остальные ячейки в этом столбце на весь экран. А если сделать его прямым потомком TableLayout, он сможет делать все, что хочет, тогда как кнопки Button останутся в своих столбцах равной ширины, расположенных по соседству друг с другом.

Учтите, что виджет TableRow не обязан объявлять атрибуты ширины и высоты. Он использует ширину и высоту, а также все остальные атрибуты TableLayout. С другой стороны, вложенный виджет LinearLayout способен обеспечить большую гибкость в определении внешнего вида ваших виджетов.

Взгляните на макет в графическом конструкторе. Какой цвет фона вы видите? В мастере для приложения HelloMoon была выбрана тема Holo Dark. Однако на момент написания книги мастер игнорировал ваш выбор и всегда назначал светлую тему. Давайте посмотрим, как решить эту проблему. (Если вы видите темный фон, значит, мастер был исправлен и вам не придется вручную сбрасывать тему приложения, как описано в следующем разделе.)

Сброс темы приложения

Тема приложения объявляется в элементе application манифеста:

```
...
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
  </application>
</manifest>
```

Атрибут android:theme не обязателен; если тема не объявлена, используется конфигурация устройства по умолчанию.

Мы видим, что заданное значение представляет собой ссылку на ресурс — @style/AppTheme. На панели Package Explorer найдите и откройте файл res/values/styles.xml. В элементе style с именем AppBaseTheme замените значение parent на android:Theme.

Листинг 13.2. Изменение файла стилей по умолчанию (res/values/styles.xml)

```
<style name="AppBaseTheme" parent="android:Theme.Light">
<style name="AppBaseTheme" parent="android:Theme">
```

В каталоге res также находятся два каталога values с уточненными именами; каждый содержит файл styles.xml. Уточнения в именах каталогов обозначают уровни API. Данные в файле res/values-11/styles.xml будут использоваться для API уровней 11–13, а значения в файле res/values-14/styles.xml — для API уровня 14 и выше.

Откройте файл res/values-11/styles.xml и замените значение атрибута `parent` у `AppBaseTheme` на `android:Theme.Holo`. Эта тема должна использоваться на всех устройствах с API 11 и выше, поэтому каталог res/values-14/ только мешает. Удалите его из проекта HelloMoon.

Сохраните файлы и снова просмотрите макет. Теперь он должен иметь темный фон, который хорошо сочетается с графическим файлом.

Создание класса HelloMoonFragment

Создайте новый класс с именем `HelloMoonFragment` и назначьте его суперклассом `android.support.v4.app.Fragment`.

Переопределите метод `HelloMoonFragment.onCreateView(…)`, чтобы заполнить только что определенный макет и получить ссылки на кнопки.

Листинг 13.3. Исходная версия HelloMoonFragment (HelloMoonFragment.java)

```
public class HelloMoonFragment extends Fragment {

    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);

        return v;
    }
}
```

Использование фрагмента макета

В приложении CriminalIntent хостинг фрагментов осуществлялся добавлением их в код активности. В приложении HelloMoon вместо этого будет использоваться фрагмент макета, при использовании которого разработчик задает класс фрагмента в элементе `fragment`.

Откройте файл activity_hello_moon.xml и замените его содержимое элементом `fragment`, приведенным в листинге 13.4.

Листинг 13.4. Создание фрагмента макета (activity_hello_moon.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>
```

Прежде чем вы сможете запустить код, необходимо внести еще одно изменение в код HelloMoonActivity. Измените суперкласс HelloMoonActivity — им должен быть класс FragmentActivity:

Листинг 13.5. Преобразование HelloMoonActivity в FragmentActivity (HelloMoonActivity.java)

```
public class HelloMoonActivity extends ~~Activity~~ FragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_moon);
    }
}
```

Запустите приложение HelloMoon. На этот раз HelloMoonActivity становится хостом представления HelloMoonFragment.

Вот и все, что необходимо для хостинга фрагмента макета. Мы указали имя класса фрагмента в макете, он был добавлен в активность и выведен на экран. А вот что при этом происходило «за кулисами»: когда класс вызвал метод .setContentView(…) и заполнил макет activity_hello_moon.xml, он обнаружил элемент fragment. В этой точке FragmentManager создал экземпляр HelloMoonFragment и добавил его в список. Затем он вызвал для HelloMoonFragment метод onCreateView(…) и поместил представление, возвращенное этим методом, в место, подготовленное тегом fragment.

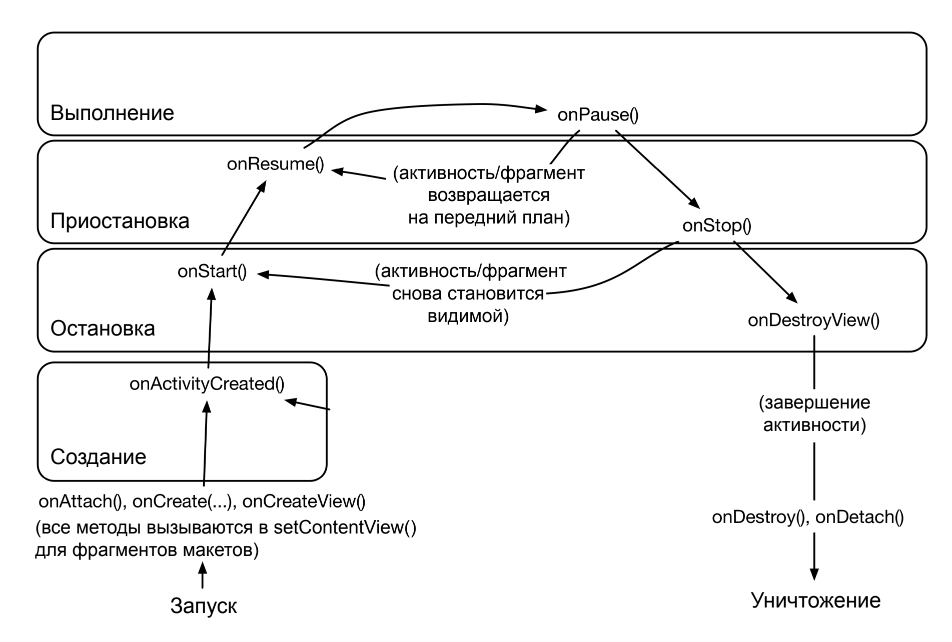


Рис. 13.5. Жизненный цикл фрагмента макета

Чем приходится расплачиваться за эту простоту? Вы теряете гибкость и широту возможностей, характерные для прямой работы с FragmentManager:

- Вы можете переопределять методы жизненного цикла фрагмента, чтобы реагировать на события, но не можете управлять тем, когда эти методы вызываются.
- Вы не можете закреплять транзакции, которые удаляют, заменяют или отсоединяют фрагмент макета. Приходится довольствоваться тем, что было сделано при создании активности.
- К фрагментам макетов нельзя присоединять аргументы. Присоединение аргументов должно осуществляться после создания фрагмента и до его включения в `FragmentManager`. С фрагментами макетов все эти события вам недоступны.

Однако в простом приложении или в статической части сложного приложения использование фрагмента макета может быть вполне разумным.

Теперь, когда мы организовали хостинг `HelloMoonFragment`, обратимся к воспроизведению аудио.

Воспроизведение аудио

Создайте в пакете `com.bignerdranch.android.hellomoon` новый класс с именем `AudioPlayer`. Оставьте его суперклассом `java.lang.Object`.

В файле AudioPlayer.java добавьте поле для хранения экземпляра `MediaPlayer` и методы для остановки и воспроизведения этого экземпляра.

Листинг 13.6. Простой код воспроизведения аудио с использованием MediaPlayer (AudioPlayer.java)

```
public class AudioPlayer {

    private MediaPlayer mPlayer;

    public void stop() {
        if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
        }
    }

    public void play(Context c) {
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
        mPlayer.start();
    }
}
```

В методе `play(Context)` вызывается метод `MediaPlayer.create(Context, int)`. Объект `Context` нужен `MediaPlayer` для опознания идентификатора ресурса аудиофайла. (Также существуют другие методы `MediaPlayer.create(…)`, используемые при получении аудио из других источников, например из Интернета или по локальному URI.)

В методе `AudioPlayer.stop()` экземпляр `MediaPlayer` освобождается, а полю `mPlayer` присваивается `null`. Вызов `MediaPlayer.release()` уничтожает экземпляр.

Уничтожение кажется слишком агрессивной интерпретацией «остановки», но для этого есть веские причины. Класс MediaPlayer удерживает аудиооборудование и другие системные ресурсы до вызова release(). Эти ресурсы совместно используются всеми приложениями. Класс MediaPlayer включает метод stop() для перевода экземпляра MediaPlayer в остановленное состояние, из которого он может быть перезапущен. Однако при простом воспроизведении аудиоданных корректнее уничтожить экземпляр вызовом release(), а потом создать его заново.

Простое правило: удерживайте ровно один экземпляр MediaPlayer и только на то время, в котором он что-то воспроизводит.

Для соблюдения этого правила мы внесем пару изменений в play(Context). Добавьте исходный вызов stop() и заставьте слушателя вызывать stop() при завершении воспроизведения.

Листинг 13.7. Только один экземпляр (AudioPlayer.java)

```
    ...
    public void play(Context c) {
        stop();

        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);

        mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
            public void onCompletion(MediaPlayer mp) {
                stop();
            }
        });

        mPlayer.start();
    }
}
```

Вызов stop() в начале play(Context) предотвращает возможное создание нескольких экземпляров MediaPlayer, если пользователь щелкнет на кнопке Play повторно. Вызов stop() при завершении воспроизведения файла освобождает экземпляр MediaPlayer, как только он перестает использоваться.

Также вызов AudioPlayer.stop() необходимо включить в HelloMoonFragment, чтобы экземпляр MediaPlayer не продолжал воспроизведение после уничтожения фрагмента. В классе HelloMoonFragment переопределите метод onDestroy() и включите в него вызов AudioPlayer.stop().

Листинг 13.8. Переопределение onDestroy() (HelloMoonFragment.java)

```
    ...

    @Override
    public void onDestroy() {
        super.onDestroy();
        mPlayer.stop();
    }
}
```

Класс MediaPlayer может продолжить воспроизведение после уничтожения HelloMoonFragment, потому что MediaPlayer работает в другом программном *потоке* (thread). Сейчас мы намеренно игнорируем этот многопоточный аспект HelloMoon. Управление потоками более подробно рассматривается в главе 26.

Подключение кнопок воспроизведения и остановки

Вернитесь к файлу HelloMoonFragment.java. Пора заняться воспроизведением аудио: создайте экземпляр класса AudioPlayer и назначьте слушателей для кнопок воспроизведения и остановки.

Листинг 13.9. Подключение кнопки Play (HelloMoonFragment.java)

```
public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);
        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.stop();
            }
        });
        return v;
    }

}
```

Запустите приложение HelloMoon, нажмите кнопку Play и насладитесь живой историей.

В этой главе мы едва затронули тему использования MediaPlayer. За информацией о возможностях MediaPlayer обращайтесь к руководству Android «MediaPlayer Developer Guide» по адресу http://developer.android.com/guide/topics/media/mediaplayer.html.

Упражнение. Приостановка воспроизведения

Предоставьте пользователю возможность приостановить воспроизведение аудио. Описания необходимых методов можно найти в справочном описании класса MediaPlayer.

Для любознательных: воспроизведение видео

В том, что касается воспроизведения видео, можно выбирать из нескольких вариантов. Можно использовать класс MediaPlayer, как мы только что сделали. Для этого достаточно подключить место для воспроизведения.

Часто обновляемые изображения (как видео) в Android отображаются на виджете SurfaceView. Точнее, они отображаются на виджете Surface, хостом которого является SurfaceView. Чтобы получить доступ к Surface, следует получить экземпляр SurfaceHolder для SurfaceView. Эта тема более подробно рассматривается в главе 19. А пока достаточно знать, что подключение SurfaceHolder к MediaPlayer осуществляется вызовом MediaPlayer.setDisplay(SurfaceHolder).

Часто для воспроизведения видео проще использовать экземпляр VideoView. Класс VideoView не взаимодействует с MediaPlayer, как SurfaceView. Однако он взаимодействует с MediaController, что позволяет легко организовать интерфейс воспроизведения.

Единственный нюанс с использованием VideoView заключается в том, что класс не принимает идентификаторы ресурсов — только пути к файлам или объекты Uri. Чтобы создать объект Uri, ссылающийся на ресурс Android, используйте код следующего вида:

```
Uri resourceUri = Uri.parse("android.resource://" +
  "com.bignerdranch.android.hellomoon/raw/apollo_17_stroll");
```

Создайте URI со схемой android.resource, именем вашего пакета вместо хоста, типом и именем вашего ресурса вместо пути. Результат может быть передан VideoView.

Упражнение. Воспроизведение видео в HelloMoon

Измените программу HelloMoon так, чтобы она также позволяла воспроизводить видеоролики. Если вы не загрузили файл apollo_17_stroll.mpg ранее, вернитесь к файлу решений и скопируйте его из проекта главы в каталог res/raw. Затем воспроизведите его одним из описанных способов.

14 Сохранение фрагментов

В настоящее время приложение HelloMoon некорректно обрабатывает повороты. Запустите его, включите воспроизведение и поверните устройство. Воспроизведение прерывается.

Дело в том, что при повороте `HelloMoonActivity` уничтожается. Когда это происходит, `FragmentManager` отвечает за уничтожение `HelloMoonFragment`. `FragmentManager` вызывает методы угасающего жизненного цикла фрагмента: `onPause()`, `onStop()` и `onDestroy()`. В `HelloMoonFragment.onDestroy()` освобождается экземпляр `MediaPlayer`, что приводит к остановке воспроизведения.

В главе 3 мы решили проблему обработки поворотов в приложении GeoQuiz переопределением `Activity.onSaveInstanceState(Bundle)`. Данные сохранялись, а новая активность загружала их. Класс `Fragment` содержит метод `onSaveInstanceState(Bundle)`, который работает аналогичным образом. Однако сохранение состояния объекта `MediaPlayer` и его последующее восстановление все равно прерывает воспроизведение и раздражает пользователей.

Сохранение фрагмента

К счастью, у фрагментов имеется механизм, благодаря которому экземпляр `MediaPlayer` может «пережить» изменение конфигурации. Переопределите метод `HelloMoonFragment.onCreate(…)` и задайте свойство фрагмента.

Листинг 14.1. Вызов setRetainInstance(true) (HelloMoonFragment.java)

```
...

private Button mPlayButton;
private Button mStopButton;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...
```

По умолчанию свойство `retainInstance` фрагмента содержит `false`. Это означает, что при поворотах фрагмент *не сохраняется*, а уничтожается и создается заново вместе с активностью-хостом. Вызов `setRetainInstance(true)` сохраняет фрагмент, который не уничтожается вместе с активностью, а передается новой активности в неизменном виде.

При сохранении фрагмента можно рассчитывать на то, что все его поля (включая `mPlayButton`, `mPlayer` и `mStopButton`) сохранят прежние значения. Вы к ним обращаетесь, а они просто находятся на своем месте.

Снова запустите HelloMoon. Включите воспроизведение, поверните устройство и убедитесь в том, что файл воспроизводится без прерывания.

Повороты и сохраненные фрагменты

Давайте поближе познакомимся с тем, как работают сохраненные фрагменты. Они используют то обстоятельство, что представление фрагмента может уничтожаться и создаваться заново без необходимости уничтожать сам фрагмент.

При изменении конфигурации `FragmentManager` сначала уничтожает представления фрагментов в своем списке. Представления фрагментов всегда уничтожаются и создаются заново по тем же причинам, по которым уничтожаются и создаются заново представления активности: в новой конфигурации могут потребоваться новые ресурсы. На случай, если для нового варианта существуют более подходящие ресурсы, представление строится «с нуля».

Затем `FragmentManager` проверяет свойство `retainInstance` каждого фрагмента. Если оно равно `false` (по умолчанию), `FragmentManager` уничтожает экземпляр фрагмента. Фрагмент и его представление будут созданы заново новым экземпляром `FragmentManager` новой активности.

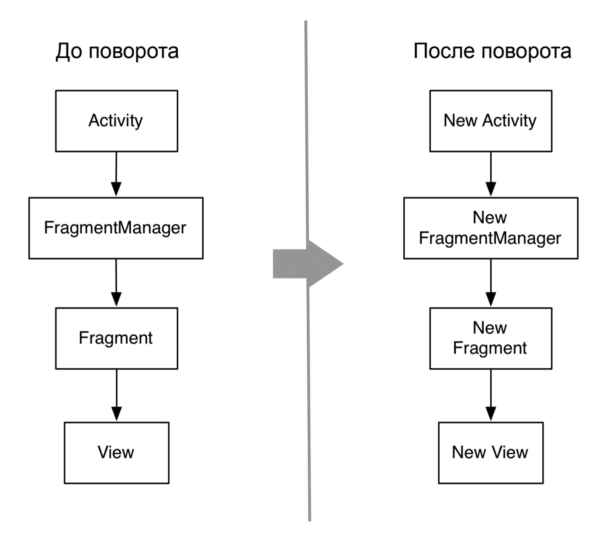


Рис. 14.1. Реализация поворота по умолчанию с UI-фрагментом

С другой стороны, если значение `retainInstance` равно `true`, представление фрагмента уничтожается, но сам фрагмент остается. При создании новой активности новый экземпляр `FragmentManager` находит сохраненный фрагмент и воссоздает его представление.

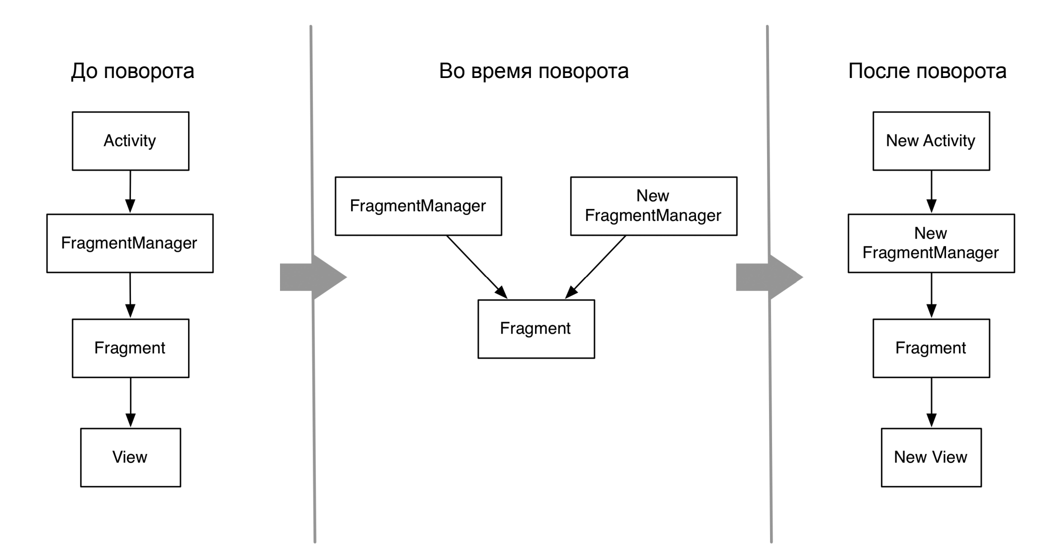


Рис. 14.2. Поворот с сохраненным UI-фрагментом

Сохраненный фрагмент не уничтожается, но *отсоединяется* (detached) от «умирающей» активности. В сохраненном состоянии фрагмент все еще существует, но не имеет активности-хоста.

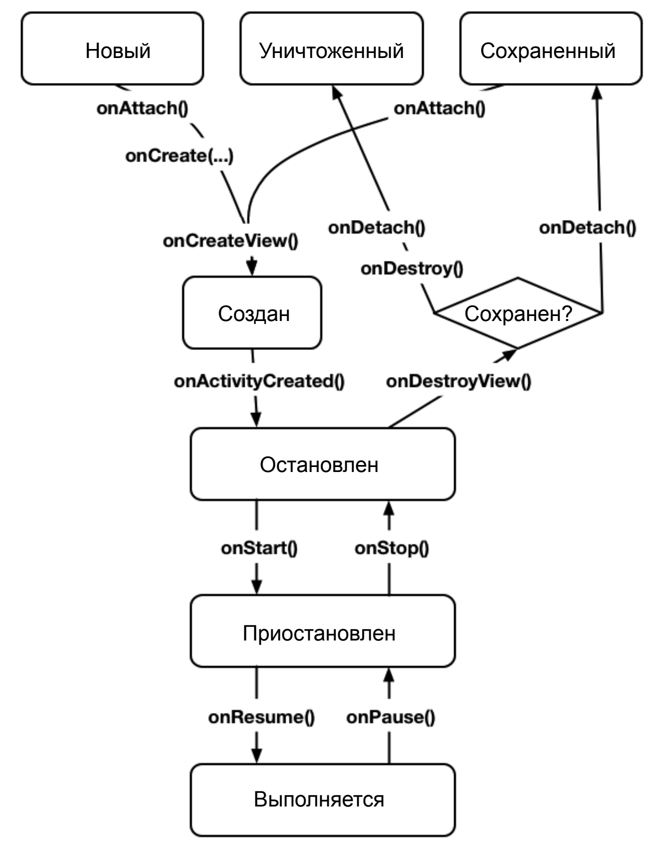


Рис. 14.3. Жизненный цикл фрагмента

Переход в сохраненное состояние происходит только при выполнении двух условий:

- для фрагмента был вызван метод `setRetainInstance(true)`;
- активность-хост уничтожается для изменения конфигурации (обычно поворот).

Фрагмент находится в сохраненном состоянии очень недолго — от момента отсоединения от старой активности до повторного присоединения к новой, немедленно создаваемой активности.

Сохранение фрагментов: действительно так хорошо?

Сохраненные фрагменты: удобно, верно? Да! Действительно удобно. На первый взгляд они решают все проблемы, связанные с уничтожением активностей и фрагментов при поворотах. При изменении конфигурации устройства для подбора наиболее подходящих ресурсов создается новое представление, а в вашем распоряжении имеется простой способ сохранения данных и объектов.

Тогда почему не сохранять все фрагменты подряд и почему фрагменты не сохраняются по умолчанию? Похоже, Android без энтузиазма относится к сохранению фрагментов в пользовательских интерфейсах. Мы не знаем, почему это так, но если группа Android ни во что не ставит эту возможность, в будущем могут возникнуть проблемы.

Помните, что сохраненные фрагменты продолжают существовать только при уничтожении активности при изменения конфигурации. Если активность уничтожается из-за того, что ОС потребовалось освободить память, то все сохраненные фрагменты также будут уничтожены.

Повороты и onSaveInstanceState(Bundle)

Метод `onSaveInstanceState(Bundle)` — еще один инструмент, используемый для обработки поворотов. Собственно, если у вашего приложения нет проблем с поворотами, то только благодаря работе поведения `onSaveInstanceState(…)` по умолчанию.

Хорошим примером служит приложение CriminalIntent. Фрагмент `CrimeFragment` не сохраняется, но если внести изменения в краткое описание преступления или переключить флажок, новые состояния объектов `View` автоматически сохраняются и восстанавливаются после поворота. Метод `onSaveInstanceState(…)` проектировался именно для решения этой задачи — сохранения и восстановления состояния пользовательского интерфейса приложения.

Главное отличие между переопределением `Fragment.onSaveInstanceState(…)` и сохранением фрагмента — продолжительность существования сохраненных данных. Если данные должны только пережить изменения конфигурации, сохранение фрагмента потребует существенно меньшей работы. Это особенно справедливо при сохранении объекта; разработчику не нужно беспокоиться о том, реализует объект `Serializable` или нет.

Но если данные должны существовать дольше, сохранение фрагмента не поможет. Если активность уничтожается для освобождения памяти после бездействия пользователя, все сохраненные фрагменты уничтожаются так же, как и их несохраненные родственники.

Чтобы разница стала более наглядной, вспомните приложение GeoQuiz. Проблема поворота заключалась в том, что индекс вопроса обнулялся при повороте. На каком бы вопросе ни находился пользователь, при повороте устройства он возвращался к первому вопросу. Чтобы пользователь видел правильный вопрос, мы сохраняли значение индекса, а затем снова загружали его.

В приложении GeoQuiz не использовались фрагменты, но представьте себе переработанную версию GeoQuiz с фрагментом `QuizFragment`, хостом которого является `QuizActivity`. Как быть — переопределить `Fragment.onSaveInstanceState(…)` для сохранения индекса или сохранить `QuizFragment` и оставить переменную живой?

На рис. 14.4 показаны три разных жизненных цикла, с которыми вам придется работать: жизнь объекта активности (и его несохраненных фрагментов), жизнь сохраненного фрагмента и жизнь записи активности.

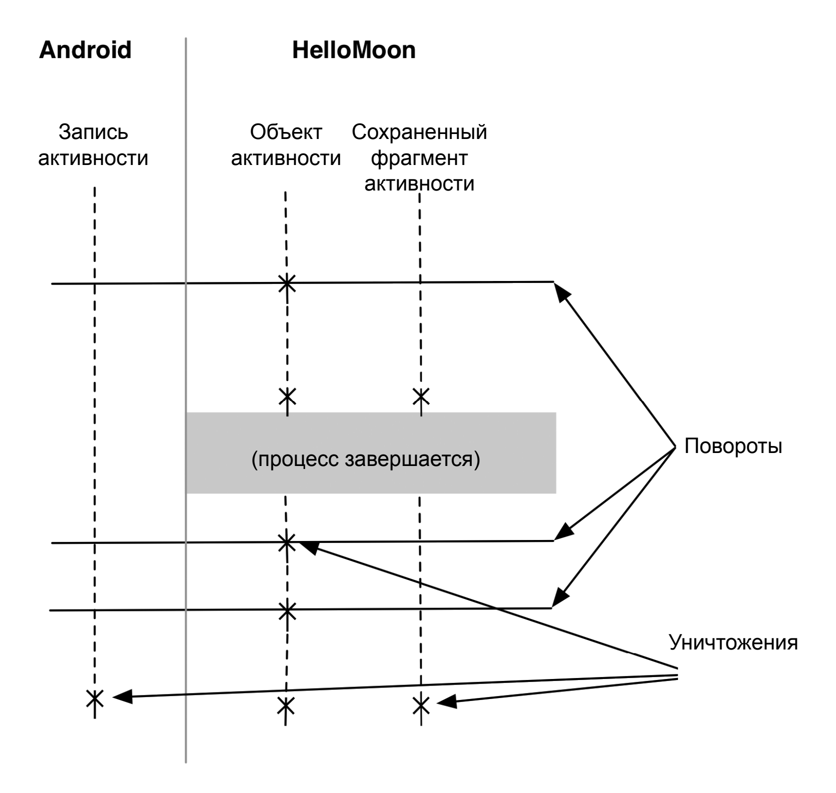


Рис. 14.4. Три жизненных цикла

Срок жизни объекта активности слишком мал; это и является источником проблемы поворота. Индекс определенно должен пережить объект активности.

Если сохранить `QuizFragment`, индекс будет существовать на протяжении срока жизни сохраненного фрагмента. Если GeoQuiz содержит только пять вопросов, сохранение `QuizFragment` проще реализуется и требует меньше кода. Мы инициализируем индекс в поле, а затем вызываем `setRetainInstance(true)` в `QuizFragment.onCreate(…)`.

Листинг 14.2. Сохранение гипотетического фрагмента QuizFragment

```
public class QuizFragment extends Fragment {
    ...
    private int mCurrentIndex = 0;
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    ...
}
```

Привязка индекса к сроку жизни сохраненного фрагмента позволяет ему пережить уничтожение объекта активности и решает проблему сброса индекса при повороте. Но как видно из рис. 14.4, сохранение `QuizFragment` не позволяет индексу пережить завершение процесса, которое может произойти, если пользователь оставит приложение на некоторое время, а активность и сохраненный фрагмент будут уничтожены для освобождения памяти.

Если список состоит всего из пяти вопросов, возвращение к началу может показаться приемлемым решением. А если бы в приложении GeoQuiz было 100 вопросов? Необходимость начинать все заново вызовет у пользователя (законное) раздражение. Состояние индекса должно переживать срок жизни записи активности. Для этого индекс будет сохраняться в `onSaveInstanceState(…)`. Затем, если пользователь оставит приложение на некоторое время, он сможет продолжить с прежнего места.

Итак, если в активности или фрагменте присутствуют данные, которые должны существовать в течение долгого времени, их следует привязать к сроку жизни записи активности, переопределяя метод `onSaveInstanceState(Bundle)` для сохранения состояния и его последующего восстановления.

Для любознательных: повороты до появления фрагментов

Фрагменты появились в версии Honeycomb и были доступны только для телефонов с Ice Cream Sandwich. С другой стороны, проблема поворота существует с самого начала.

Чтобы оценить всю прелесть сохранения фрагментов, нужно хотя бы примерно представлять, что происходило до их появления.

- Хотите сохранить объект между изменениями конфигурации? Для этого использовался метод с именем `onRetainNonConfigurationInstance()`. Разработчик переопределял метод `onRetainNonConfigurationInstance()`, чтобы он возвращал сохраняемый объект. Чтобы получить объект обратно, вызывался метод `getLastNonConfigurationInstance()`.
- Хотите сохранить в активности более одного объекта? Что ж, вам не повезло. Каждая активность может хранить только один объект. Если вам этого мало, придется возиться с упаковкой нескольких объектов в один.
- Используете объект (например, `MediaPlayer`), который должен освобождаться при уничтожении активности? Придется проверять, что метод `onRetainNonConfigurationInstance()` не вызывался, чтобы вы могли выполнить нужные действия в `onDestroy()`.

Метод `onRetainNonConfigurationInstance()` сейчас считается устаревшим, так что передача существующих объектов при поворотах теперь всегда осуществляется через сохранение фрагментов. Если вы уже немного освоились с использованием фрагментов, вы обнаружите, что этот способ намного приятнее старого.

15 Локализация

Локализацией (localization) называется процесс выбора ресурсов приложения в зависимости от конфигурации языка устройства. В этой главе мы локализуем приложение HelloMoon — в частности, будут определены испанские версии звукового и строкового файла. Если на устройстве выбран испанский язык, Android находит и использует испанские ресурсы.

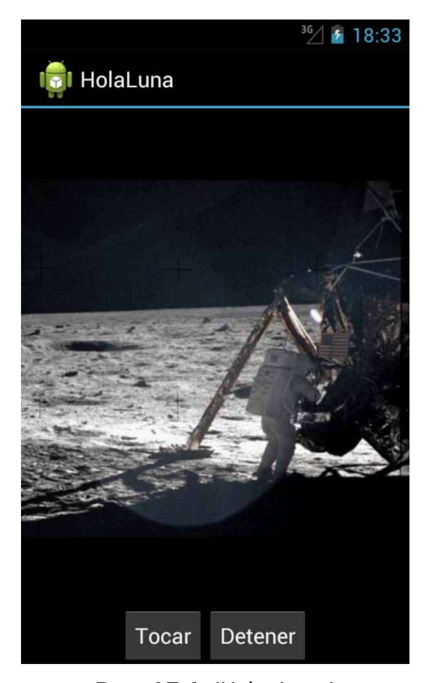


Рис. 15.1. ¡Hola, Luna!

Локализация ресурсов

Языковые настройки являются частью конфигурации устройства. Android предоставляет конфигурационные квалификаторы для разных языков, существенно упрощающие локализацию: вы создаете подкаталоги ресурсов с разными языковыми квалификаторами и размещаете в них альтернативные ресурсы. Система ресурсов Android делает все остальное.

Языковые квалификаторы позаимствованы из кодов ISO 639-1. Испанскому языку соответствует квалификатор `-es`. Создайте в проекте HelloMoon два новых подкаталога res: res/raw-es/ и res/values-es/.

Ресурсы для испанского языка включены в файл решений этой книги (http://www.bignerdranch.com/solutions/AndroidProgramming.zip). Найдите в этом архиве файлы:

- 15_Localization/HelloMoon/res/raw-es/one_small_step.wav
- 15_Localization/HelloMoon/res/values-es/strings.xml

Скопируйте их в соответствующие каталоги из только что созданных.

Вот и все, что нужно сделать, чтобы предоставить локализованные ресурсы для вашего приложения. Чтобы убедиться в этом, переключите язык устройства на испанский (откройте панель Settings и найдите языковые настройки; в зависимости от версии Android раздел будет называться Language and input, Language and Keyboard или что-нибудь в этом роде.

Добравшись до списка языковых настроек, выберите испанский язык (Español). Регион (España или Estados Unidos) не играет роли, потому что каталоги ресурсов содержат квалификатор `-es`, подходящий для обоих вариантов.

(Каталог ресурсов можно уточнить квалификатором «язык+регион» для более точного определения ресурсов. Например, для испанского языка в Испании используется квалификатор `-es-rES`, где `r` обозначает квалификатор региона, а `ES` — код Испании по стандарту ISO 3166-1-alpha-2. Обратите внимание: регистр символов в конфигурационных квалификаторах не учитывается, но рекомендуется соблюдать стандартную схему Android: код языка записывается символами нижнего регистра, а код региона — символами верхнего регистра с префиксом `r` (в нижнем регистре).

Запустите приложение HelloMoon, нажмите кнопку Tocar и послушайте, как знаменитые слова Нила Армстронга звучат на испанском языке.

Ресурсы по умолчанию

Для английского языка используется конфигурационный квалификатор `-en`. Возможно, в порыве локализационного усердия вам захочется переименовать существующие каталоги raw и values в raw-en/ и values-en/.

Это не лучшая мысль. Ресурсы в этих каталогах являются *ресурсами по умолчанию*. Каталоги ресурсов по умолчанию не имеют квалификаторов. Наличие ресурсов по умолчанию играет важную роль: если Android не удается найти ресурс, подходящий для конфигурации устройства, при их отсутствии в приложении произойдет сбой.

Например, если файл strings.xml присутствует в каталогах values-en/ и values-es/, но не в каталоге values/, при попытке запустить приложение HelloMoon на устройстве с любым языком, кроме испанского и английского, оно аварийно завершится. Наличие ресурсов по умолчанию обеспечивает безопасность приложения независимо от конфигурации устройства.

Плотность пикселов и ресурсы по умолчанию

Единственным исключением из правила предоставления ресурсов по умолчанию является экранная плотность пикселов. Имена каталогов drawable в проекте обычно уточняются для разных плотностей экрана квалификаторами -mdpi, -hdpi и -xhdpi. Однако выбор ресурсов drawable в Android не сводится к простому подбору каталога для плотности пикселов устройства или использованию каталога по умолчанию при отсутствии совпадения.

Выбор определяется сочетанием размера экрана и плотности, и Android может выбрать ресурс из каталога с квалификатором меньшей или большей плотности с последующим масштабированием. Более подробная информация приведена в документации по адресу http://developer.android.com/guide/practices/screens_support.html, но важный момент заключается в том, что размещать графические ресурсы по умолчанию в res/drawable/ не обязательно.

Конфигурационные квалификаторы

Вы уже видели и использовали конфигурационные квалификаторы для выбора альтернативных ресурсов: языка (например, values-es/), ориентации экрана (например, layout-land/), плотности экрана (например, drawable-mdpi/) и уровня API (например, values-v11/).

В табл. 15.1 перечислены характеристики конфигурации устройств с конфигурационными квалификаторами, распознаваемыми системой Android для выбора ресурсов.

Таблица 15.1. Характеристики с конфигурационными квалификаторами

Мобильный код страны (MCC) с необязательным мобильным кодом сети (MNC)
Код языка с необязательным кодом региона
Направление макета
Наименьшая ширина
Доступная ширина
Доступная высота
Размер экрана
Соотношение сторон экрана
Ориентация экрана

продолжение ⇗

Таблица 15.1 (*продолжение*)

Режим пользовательского интерфейса
Ночной режим
Плотность экрана (dpi)
Тип сенсорного экрана
Доступность клавиатуры
Основной метод ввода текста
Доступность навигационных клавиш
Основной нетекстовый метод навигации
Уровень API

Описания этих характеристик и примеры конкретных конфигурационных квалификаторов можно найти по адресу *http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources*.

Приоритеты альтернативных ресурсов

При таком количестве конфигурационных квалификаторов для выбора ресурсов возможны ситуации, в которых конфигурация устройства соответствует сразу нескольким альтернативным ресурсам. В таких ситуациях квалификаторы рассматриваются в порядке, представленном в табл. 15.1.

Чтобы понять, как работает система приоритетов, мы добавим в HelloMoon еще один альтернативный ресурс — версию строкового ресурса app_name для альбомной ориентации. Ресурс app_name используется как заголовок активности — этот текст выводится на панели действий. В альбомной ориентации появляется больше места для вывода текста, и мы можем воспользоваться этим обстоятельством.

Создайте каталог values-land и скопируйте в него строковый файл по умолчанию.

Единственный строковый ресурс, который должен измениться в альбомной ориентации, — app_name. Удалите другие строковые ресурсы и измените значение app_name так, как показано в листинге 15.1.

Листинг 15.1. Создание альтернативного строкового ресурса для альбомной ориентации (values-land/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Hello, Moon! How are you? Is it cold up there?</string>
  <string name="app_name">HelloMoon</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="hellomoon_play">Play</string>
  <string name="hellomoon_stop">Stop</string>
  <string name="hellomoon_image_description">Neil Armstrong stepping
```

```
        onto the moon</string>
</resources>
```

Альтернативы для строковых ресурсов (и других ресурсов values) подбираются на уровне отдельных строк, так что в случае совпадения строки дублировать не обязательно. Дубликаты только усложнят сопровождение приложения с течением времени.

Теперь у нас имеются три версии `app_name`: версия по умолчанию в файле values/strings.xml, испанская альтернатива в файле values-es/strings.xml и альтернатива для альбомной ориентации в файле values-land/strings.xml.

Включите на своем устройстве испанский язык, запустите HelloMoon и поверните устройство. Альтернатива для испанского языка обладает более высоким приоритетом, поэтому вы увидите строку из файла values-es/strings.xml.

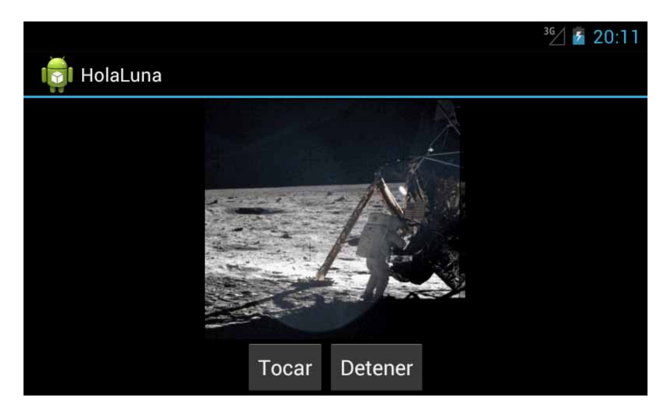


Рис. 15.2. Android отдает предпочтение языку перед ориентацией экрана

Если хотите, попробуйте вернуться к английскому языку, снова запустить приложение и убедиться в том, что на экране появляется альтернативная строка для альбомной ориентации (как и предполагалось). Чтобы вернуться к английскому языку, поищите в лаунчере раздел Ajustes или Configuración и найдите параметр со словом idioma (язык).

Множественные квалификаторы

Каталогу ресурсов можно назначить более одного квалификатора. Например, чтобы задать в приложении HelloMoon строку на испанском языке для альбомной ориентации, создайте каталог с именем values-es-land.

Если каталог имеет более одного квалификатора, они должны следовать в порядке приоритетов. Таким образом, values-es-land является допустимым именем каталога, а values-land-es — нет.

(Учтите, что квалификатор «язык-регион» — например, `-es-rES` — выглядит как объединение двух разных квалификаторов, но это впечатление обманчиво. Обозначение региона само по себе не является действительным квалификатором.)

Скопируйте файл values-land/strings.xml в каталог values-es-land/ и внесите изменения, представленные в листинге 15.2 (или, если хотите, скопируйте файл res/values-es-land/strings.xml из проекта решений).

Листинг 15.2. Создание испанского строкового ресурса для альбомной ориентации (values-es-land/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Hello, Moon! How are you? Is it cold up there?</string>
  <string name="app_name">¡Hola, Luna! ¿Como estás? ¿Hace frío ahí arriba?</string>
</resources>
```

Выберите испанский язык, запустите HelloMoon и убедитесь в том, что новый альтернативный ресурс появляется в положенном месте.



Рис. 15.3. Испанская строка для альбомного режима отображается на экране

Поиск наиболее подходящих ресурсов

Давайте разберемся подробнее, как же Android определяет, какую версию `app_name` следует отображать при запуске. Сначала Android рассматривает три варианта строкового ресурса с именем `app_name`:

- values-es/strings.xml
- values-land/strings.xml
- values-es-land/strings.xml

Затем рассматривается конфигурация устройства. В настоящее время она включает испанский язык и альбомную ориентацию экрана.

Исключение несовместимых каталогов

Поиск оптимального ресурса Android начинает с исключения всех каталогов ресурсов, несовместимых с текущей конфигурацией.

Ни один из трех вариантов не является несовместимым с текущей конфигурацией. Если повернуть устройство в книжную ориентацию, конфигурация изменится, а каталоги ресурсов values-land/ и values-es-land/ станут несовместимыми и будут исключены.

С некоторыми квалификаторами все не так однозначно в отношении совместимости конфигураций. Например, уровень API не требует жесткого совпадения. Квалификатор `-v11` совместим со всеми устройствами с уровнем API 11 и выше.

Некоторые квалификаторы были добавлены на более поздних уровнях API. Например, квалификаторы направления макета появились в API уровня 17; два допустимых значения — `-ldltr` (слева направо) и `-ldrtl` (справа налево). Новые квалификаторы сопровождаются неявным квалификатором уровня API, так что `-ldltr` можно рассматривать как `-ldltr-v17`. (Добавление скрытых квалификаторов — еще одна веская причина для включения ресурсов по умолчанию.)

С плотностью экрана проблема решается иначе, без учета совместимости. Android выбирает ресурс, который считается наиболее подходящим для конфигурации; он может быть (а может и не быть) ресурсом, квалификаторы которого соответствуют конфигурации.

Перебор таблицы приоритетов

После исключения несовместимых каталогов Android начинает перебирать таблицу приоритетов (см. табл. 15.1), начиная с самого высокоприоритетного квалификатора MCC. При наличии каталога ресурсов с квалификатором MCC все каталоги ресурсов, *не имеющие* квалификатора MCC, исключаются. Если после этого остается более одного подходящего каталога, Android рассматривает следующий по приоритету квалификатор и продолжает до тех пор, пока не останется всего один каталог.

В нашем примере каталогов с квалификатором MCC нет, поэтому ни один каталог не исключается, а Android переходит к следующему пункту — квалификатору языка. Два каталога (values-es/ и values-es-land/) содержат квалификаторы языка. У одного каталога (values-land/) такого квалификатора нет, и он исключается.

Android продолжает опускаться по списку квалификаторов. Достигнув ориентации экрана, Android находит один каталог с квалификатором ориентации и один каталог без него. Каталог values-es/ исключается, и остается только каталог values-es-land/. Таким образом, Android использует ресурс из values-es-land/.

Дополнительные правила использования ресурсов

Теперь, когда вы лучше разбираетесь в системе ресурсов Android, мы представим некоторые системные требования, которые также следует учитывать при построении собственных приложений.

Имена ресурсов

Имена ресурсов должны записываться символами нижнего регистра и не могут содержать пробелов: one_small_step.wav, app_name, armstrong_on_moon.jpg.

Ссылки на ресурсы в XML или коде не включают расширение файла. Например, ссылка в файле макета имеет вид @drawable/armstrong_on_moon, а ссылка в коде — R.drawable.armstrong_on_moon. Следовательно, вы не сможете различать одноименные файлы ресурсов, находящиеся в одном каталоге, по расширениям.

Структура каталогов ресурсов

Ресурсы должны храниться в одним из подкаталогов res/. Сохранение ресурсов в корневом каталоге res/ запрещено, это приведет к ошибкам при построении приложения.

Имена подкаталогов res напрямую связаны с процессом построения Android и не могут изменяться. Мы уже видели каталоги drawable/, layout/, menu/, raw/ и values/. Полный список поддерживаемых подкаталогов res (без квалификаторов) находится по адресу http://developer.android.com/guide/topics/resources/providing-resources.html#ResourceTypes.

Все остальные подкаталоги, создаваемые в res/, игнорируются. Создание каталога res/my_stuff не приведет к ошибке, но Android не будет использовать ресурсы из этого каталога.

Кроме того, в res/ нельзя создавать дополнительные уровни подкаталогов. Это ограничение иногда осложняет жизнь разработчикам. Реальные проекты могут содержать сотни графических ресурсов; желание распределить их по подкаталогам вполне естественно. Однако делать это нельзя. Единственное, что поможет вам в этой ситуации, — тщательный выбор имен ресурсов, чтобы порядок их сортировки упрощал поиск конкретных файлов.

Примером такой стратегии сортировки служит схема выбора имен файлов макетов в Android. Файлы макетов начинаются с типа представления, определяемого макетом (например, activity_, dialog_ или list_item_). Например, в каталоге res/layout/ приложения CriminalIntent содержатся файлы activity_crime_pager, activity_fragment, dialog_date, fragment_crime и list_item_crime. Макеты активностей группируются в порядке сортировки, что теоретически упрощает их поиск.

Тестирование альтернативных ресурсов

Макеты и другие ресурсы необходимо тестировать. По тестовым макетам вы узнаете, сколько альтернатив вам понадобится для разных размеров экранов, ориентаций и т. д. Тестирование может осуществляться как на реальных, так и на виртуальных устройствах, причем вы также можете использовать графический конструктор макетов.

Графический конструктор макетов содержит много инструментов для предварительного просмотра макета в разных конфигурациях. Макет можно просматривать для разных размеров экранов, типов устройств, уровней API, языков и т. д.

Чтобы просмотреть эти варианты, откройте файл fragment_hello_moon.xml в графическом конструкторе макетов. Опробуйте инструменты, обозначенные на следующем рисунке.

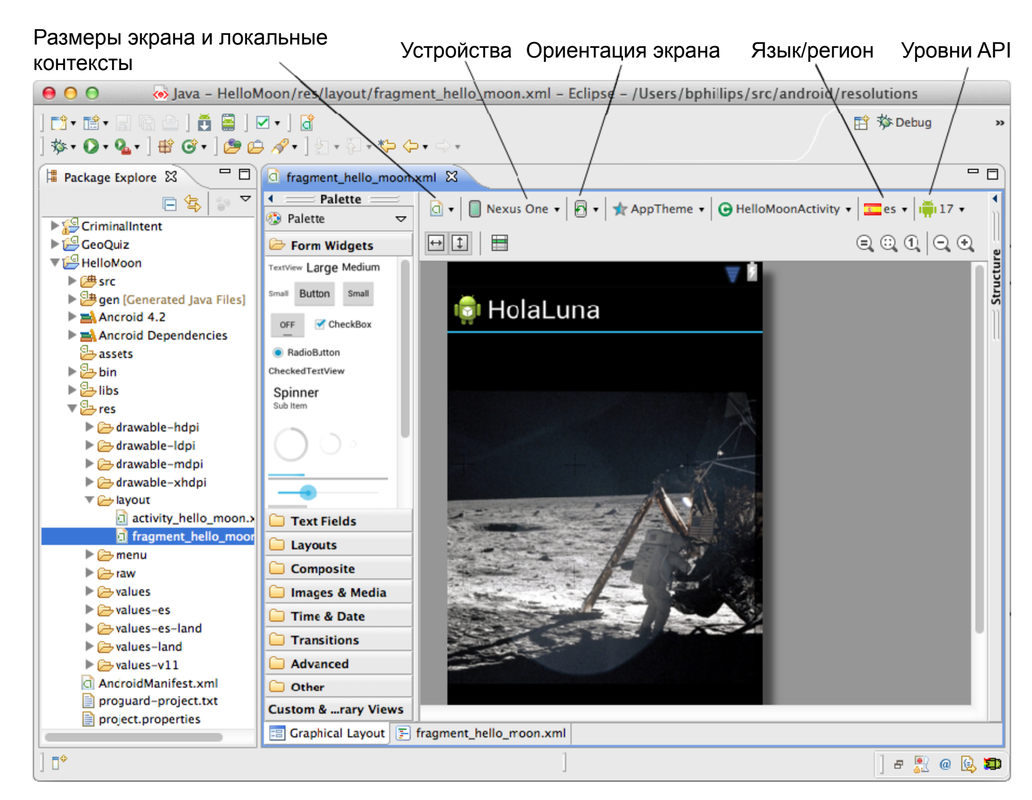


Рис. 15.4. Предварительный просмотр макета в графическом конструкторе

Чтобы убедиться в том, что вы включили все необходимые ресурсы по умолчанию, назначьте приложению язык, для которого не были заданы локализованные ресурсы. Запустите приложение, посетите все представления и проверьте повороты. Если в приложении произойдет сбой, поищите в LogCat сообщение «Resource not found...» и определите отсутствующий ресурс по умолчанию.

Прежде чем переходить к следующей главе, верните на своем устройстве основной язык вместо испанского. Для этого поищите в лаунчере раздел Ajustes или Configuración и найдите параметр со словом idioma (язык).

16 Панель действий

Панель действий (action bar) появилась в Honeycomb. Она заменяет классическую строку заголовка, но ее функции не сводятся к простому отображению значка приложения и заголовка. На панели действий также могут размещаться команды меню, а значок приложения может использоваться как навигационная кнопка.

В этой главе мы создадим меню для приложения CriminalIntent. В меню будет присутствовать *команда меню* для добавления нового преступления. Кроме того, кнопка приложения будет выполнять функции кнопки Up.

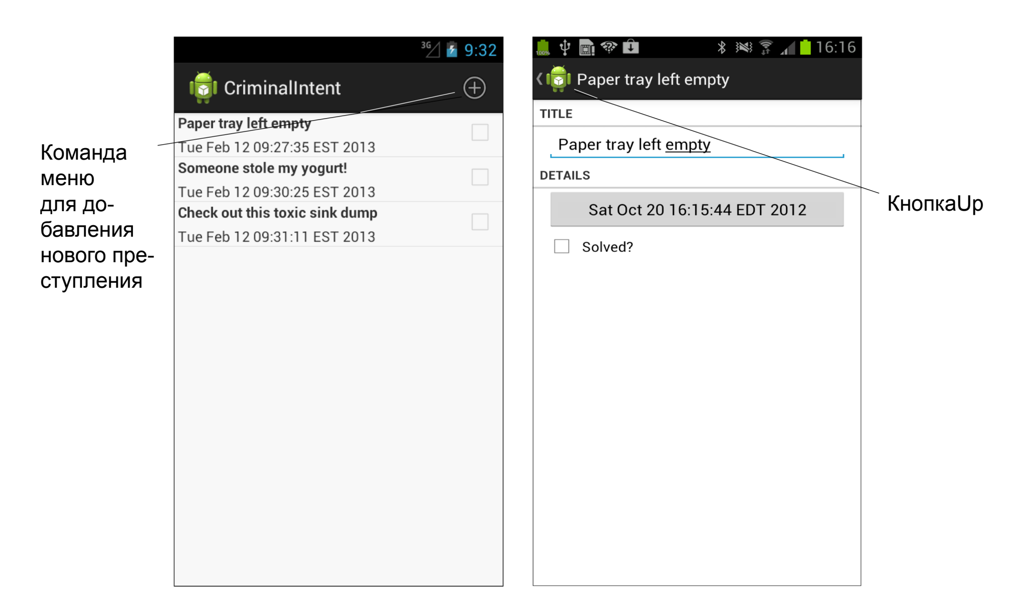


Рис. 16.1. Расширенные возможности панели действий

Командное меню

Меню на панели действий называется *командным меню* (options menu). Пункты этого меню относятся к экрану или приложению в целом. Хорошим примером служит добавление нового преступления. С другой стороны, операция удаления преступления из списка лучше подходит для *контекстного меню*, поскольку для нее необходим контекст — информация о том, какое преступление нужно удалить. Контекстные меню более подробно рассматриваются в главе 18.

Нашему командному меню потребуются новые строковые ресурсы. Один ресурс также понадобится и для контекстного меню в главе 18. Добавьте строковые ресурсы для обеих глав в файл strings.xml (листинг 16.1). Пока эти строки выглядят довольно загадочно, но лучше решить эту проблему сразу. Когда они понадобятся нам позднее, они уже будут на своем месте, и нам не придется отвлекаться от текущих дел.

Листинг 16.1. Добавление строк для меню (res/values/strings.xml)

```
  ...
  <string name="crimes_title">Crimes</string>
  <string name="crime_date_label">Date:</string>
  <string name="date_picker_title">Date of crime:</string>
  <string name="new_crime">New Crime</string>
  <string name="show_subtitle">Show Subtitle</string>
  <string name="hide_subtitle">Hide Subtitle</string>
  <string name="subtitle">If you see something, say something.</string>
  <string name="delete_crime">Delete</string>
</resources>
```



Рис. 16.2. Командные меню до Honeycomb

Панель действий для размещения командных меню используется относительно недавно, но сами командные меню существуют с первых версий Android.

К счастью, с командными меню почти не бывает проблем совместимости. Код один и тот же, а каждое устройство выбирает способ представления командных меню в зависимости от уровня Android. В этой главе мы еще вернемся к проблемам совместимости, командных меню и панели действий.

Определение командного меню в XML

Меню определяются такими же ресурсами, как и макеты. Вы создаете описание меню в XML и помещаете файл в каталог res/menu своего проекта. Android генерирует идентификатор ресурса для файла меню, который затем используется для заполнения меню в коде.

На панели Package Explorer создайте в каталоге res/ подкаталог menu. Щелкните правой кнопкой мыши на новом каталоге и выберите команду New ▸ Android XML File. Убедитесь в том, что для создаваемого файла выбран тип ресурса Menu, и присвойте файлу имя fragment_crime_list.xml.

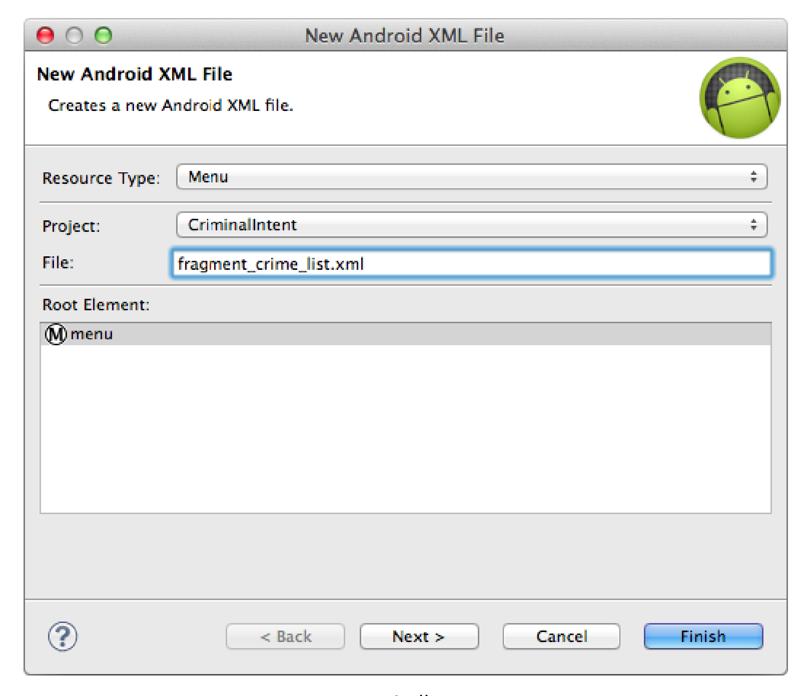


Рис. 16.3. Создание файла командного меню

Откройте файл fragment_crime_list.xml и перейдите к XML. Добавьте элемент `item`, представленный в листинге 16.2.

Листинг 16.2. Создание ресурса меню CrimeListFragment (fragment_crime_list.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/new_crime"
    android:showAsAction="ifRoom|withText"/>
</menu>
```

Атрибут `showAsAction` показывает, должна ли команда меню отображаться на самой панели действий или в *дополнительном меню* (overflow menu). Мы объединили два значения `ifRoom` и `withText`, чтобы при наличии свободного места на панели действий отображался значок и текст команды. Если на панели не хватает места для текста, то отображается только значок. Если места нет ни для того, ни для другого, команда перемещается в дополнительное меню.

Способ обращения к дополнительному меню зависит от устройства. Если у устройства имеется физическая клавиша вызова меню, то для вызова дополнительного меню необходимо нажать ее. Большинство новых устройств не имеет физической клавиши меню, поэтому дополнительное меню вызывается значком в виде трех точек в правой части панели действий (рис. 16.4).

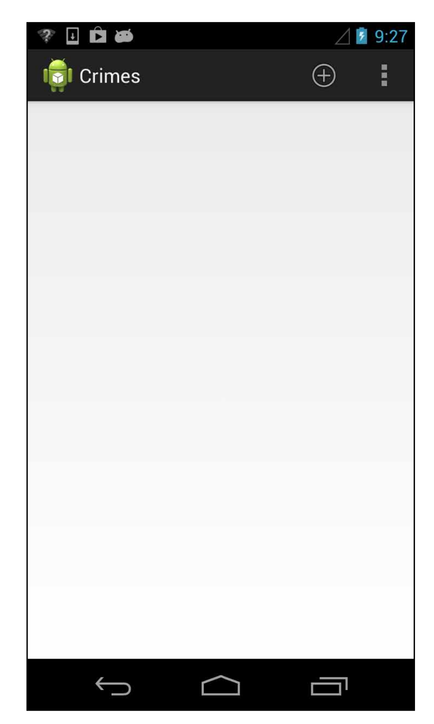


Рис. 16.4. Дополнительное меню на панели действий

Также атрибут `showAsAction` может принимать значения `always` и `never`. Выбирать `always` не рекомендуется; лучше использовать `ifRoom` и предоставить решение ОС. Вариант `never` хорошо подходит для редко выполняемых действий. Как правило, на панели действий следует размещать только часто используемые команды меню, чтобы не загромождать экран.

Учтите, что Android Lint не жалуется на атрибут `android:showAsAction`, появившийся в API уровня 11. Для атрибутов XML такая защита, как для кода Java, не нужна. Разметка XML будущих версий просто игнорируется в более ранних API.

В атрибуте `android:icon` значение `@android:drawable/ic_menu_add` ссылается на *системный значок* (system icon). Системные значки находятся на устройстве, а не в ресурсах проекта.

Использование системных значков

В прототипе ссылки на системные значки работают нормально. Однако в приложении, готовом к выпуску, лучше быть уверенным в том, что именно пользователь увидит на экране. Системные значки могут сильно различаться между устройствами и версиями ОС, а на некоторых устройствах системные значки могут не соответствовать дизайну приложения.

Одно из возможных решений — создание собственных значков. Вам придется подготовить версии для каждого разрешения экрана, а возможно, и для других конфигураций устройств. За дополнительной информацией обращайтесь к руководству «Android's Icon Design Guidelines» по адресу http://developer.android.com/guide/practices/ui_guidelines/icon_design.html.

Также можно действовать иначе — найти системные значки, соответствующие потребностям вашего приложения, и скопировать их прямо в графические ресурсы проекта. Это простой способ снабдить приложение хорошо знакомыми и понятными значками.

Чтобы найти системные значки, перейдите к каталогу установки Android SDK и найдите путь вида *домашний-каталог-android-SDK*/platforms/*уровень-API*/data/res. Например, каталог Android 4.2 на одном из наших Mac имеет вид /Developer/android-sdk-mac_86/platforms/android-17/data/res.

Просмотрите системные значки для разных SDK или проведите поиск по условию `ic_menu_add`. Вы можете скопировать результаты в подходящую папку drawable ресурсов вашего проекта. Используйте в файле макета атрибут `icon` вида `android:icon="@drawable/ic_menu_add"`, чтобы значок извлекался из ресурсов проекта, а не из системных значков устройства.

Создание командного меню

Для управления командными меню в коде используются методы обратного вызова класса `Activity`. Когда возникает необходимость в командном меню, Android вызывает метод `Activity` с именем `onCreateOptionsMenu(Menu)`.

Однако архитектура нашего приложения требует, чтобы реализация находилась в фрагменте, а не в активности. Класс `Fragment` содержит собственный набор методов обратного вызова для командных меню, которые мы реализуем в CrimeListFragment. Для создания командного меню и обработки выбранных команд используются следующие методы:

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

В файле CrimeListFragment.java переопределите метод `onCreateOptionsMenu(Menu, MenuInflater)` так, чтобы он заполнял меню, определенное в файле fragment_crime_list.xml.

Листинг 16.3. Заполнение командного меню (CrimeListFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
```

В этом методе мы вызываем метод `MenuInflater.inflate(int, Menu)` и передаем идентификатор ресурса своего файла меню. Вызов заполняет экземпляр `Menu` командами, определенными в файле.

Обратите внимание на вызов реализации `onCreateOptionsMenu(…)` суперкласса. Он не обязателен, но мы рекомендуем вызывать версию суперкласса просто для соблюдения общепринятой схемы, чтобы работала вся функциональность командных меню, определяемая в суперклассе. Впрочем, в данном случае это лишь формальность — базовая реализация этого метода из `Fragment` не делает ничего.

`FragmentManager` отвечает за вызов `Fragment.onCreateOptionsMenu(Menu, MenuInflater)` при получении активностью обратного вызова `onCreateOptionsMenu(…)` от ОС. Вы должны явно указать `FragmentManager`, что фрагмент должен получить вызов `onCreateOptionsMenu(…)`. Для этого вызывается следующий метод:

```
public void setHasOptionsMenu(boolean hasMenu)
```

В методе `CrimeListFragment.onCreate(…)` сообщите `FragmentManager`, что экземпляр `CrimeListFragment` должен получать обратные вызовы командного меню.

Листинг 16.4. Вызов hasOptionsMenu (CrimeListFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);

    getActivity().setTitle(R.string.crimes_title);
    ...
```

В приложении CriminalIntent появляется командное меню (рис. 16.5).

Где текст командного меню? У большинства телефонов в книжной ориентации хватает места только для значка. Текст команды открывается долгим нажатием на значке на панели действий.

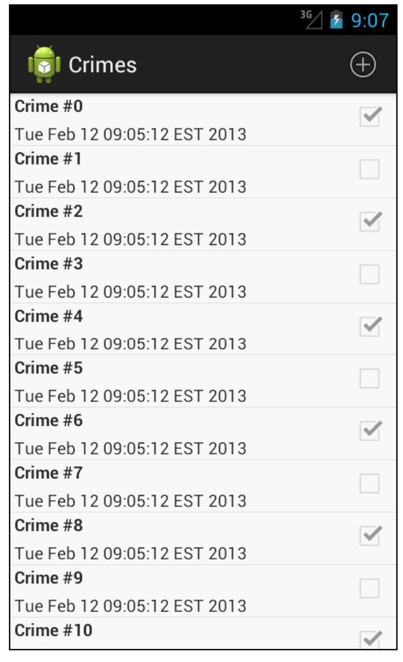


Рис. 16.5. Значок команды меню на панели действий



Рис. 16.6. Долгое нажатие на значке на панели действий выводит текст команды

В альбомной ориентации на панели действий хватает места как для значка, так и для текста (рис. 16.7).

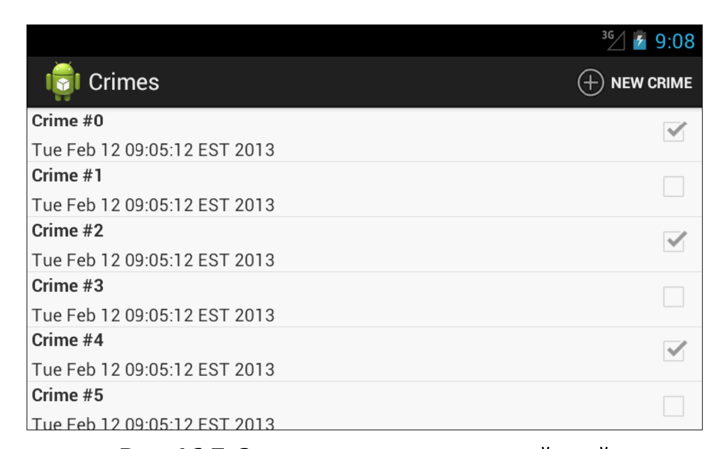


Рис. 16.7. Значок и текст на панели действий

Если запустить CriminalIntent на устройстве до Honeycomb, для просмотра командного меню следует нажать физическую клавишу меню на устройстве. Меню появляется в нижней части экрана. На рис. 16.8 изображен результат выполнения тог же кода на устройстве Gingerbread.



Рис. 16.8. Командное меню на устройстве Gingerbread

Код командного меню нормально работает и на старых, и на новых устройствах; вам не придется заниматься упаковкой и защитной проверкой версий. Однако существует небольшая разница в поведении меню в новых и старых SDK. На устройствах с версией Honeycomb и выше вызов метода `onCreateOptionsMenu(…)` и создание командного меню происходят при запуске активности. Меню должно быть готово к началу жизненного цикла активности, чтобы команды меню появились на панели действий. На старых устройствах вызов `onCreateOptionsMenu(…)` и создание меню выполняются при первом нажатии пользователем клавиши меню.

(Возможно, вы видели на старых устройствах приложения с панелью действий. Обычно такие приложения используют стороннюю библиотеку `ActionBarSherlock` для имитации функциональности панели действий на более ранних уровнях API. Библиотека `ActionBarSherlock` более подробно рассматривается в конце главы 18.)

Реакция на выбор команд

Чтобы отреагировать на выбор пользователем команды New Crime, нам понадобится механизм добавления нового объекта `Crime` в список. Включите в файл CrimeLab.java следующий метод.

Листинг 16.5. Добавление нового объекта Crime (CrimeLab.java)

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}

public ArrayList<Crime> getCrimes() {
    return mCrimes;
}

...
```

Теперь, когда вы сможете вводить описания преступлений самостоятельно, программное генерирование 100 объектов становится лишним. В файле CrimeLab.java удалите код, генерирующий эти преступления.

Листинг 16.6. Долой случайные преступления! (CrimeLab.java)

```
public CrimeLab(Context appContext) {
    mAppContext = appContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setDate(new Date());
        c.setSolved(i % 2 == 0); // Каждое второе
        mCrimes.add(c);
    }
}
```

Когда пользователь выбирает команду в командном меню, фрагмент получает обратный вызов метода `onOptionsItemSelected(MenuItem)`. Этот метод получает экземпляр `MenuItem`, описывающий выбор пользователя.

И хотя наше меню состоит всего из одной команды, в реальных меню обычно больше. Чтобы определить, какая команда меню была выбрана, проверьте идентификатор команды меню и отреагируйте соответствующим образом. Этот идентификатор соответствует идентификатору, назначенному команде в файле меню.

В файле CrimeListFragment.java реализуйте метод `onOptionsItemSelected(MenuItem)`, реагирующий на выбор команды меню. Реализация создает новый объект `Crime`, добавляет его в `CrimeLab` и запускает экземпляр `CrimePagerActivity` для редактирования нового объекта `Crime`.

Листинг 16.7. Реакция на выбор команды меню (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

```
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
            startActivityForResult(i, 0);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Метод возвращает логическое значение. После того как команда меню будет обработана, верните `true`; тем самым вы сообщаете, что дальнейшая обработка не нужна. Секция `default` вызывает реализацию суперкласса, если идентификатор команды не известен в вашей реализации.

Запустите приложение CriminalIntent и опробуйте новую команду. Добавьте несколько преступлений и отредактируйте их.

(Пустой список до начала ввода может сбить с толку пользователя. Упражнение в конце этой главы будет выдавать подсказку для пользователя при пустом списке.)

Включение иерархической навигации

До настоящего момента приложение CriminalIntent использует кнопку Back для навигации по приложению. Кнопка Back возвращает приложение к предыдущему состоянию. С другой стороны, *иерархическая навигация* осуществляет перемещение по иерархии приложения.

Android позволяет легко использовать значок приложения на панели действий для иерархической навигации. Он может осуществлять возврат к исходному экрану приложения через всю иерархию (изначально значок назывался «кнопкой Home» приложения). Однако сейчас Android рекомендует реализовать значок приложения так, чтобы переход осуществлялся на один уровень «наверх» к родителю текущей активности. При соблюдении этой рекомендации значок становится «кнопкой Up»).

В этом разделе мы реализуем функциональность кнопки Up для значка приложения на панели действий `CrimePagerActivity`. Нажимая значок, пользователь будет возвращаться к списку преступлений.

Включение значка приложения

Чтобы сообщить, что значок приложения выполняет функции кнопки Up, разработчик обычно отображает слева от него треугольную стрелку, указывающую влево, как на рис. 16.9.

Чтобы значок приложения работал как кнопка, а в представлении фрагмента отображалась стрелка, следует установить свойство фрагмента вызовом следующего метода:

```
public abstract void setDisplayHomeAsUpEnabled(boolean showHomeAsUp)
```

Этот метод относится к API уровня 11, поэтому его следует заключить в конструкцию проверки, чтобы приложение было совместимым с Froyo и Gingerbread. Также метод следует пометить аннотацией, чтобы отменить предупреждения Android Lint.

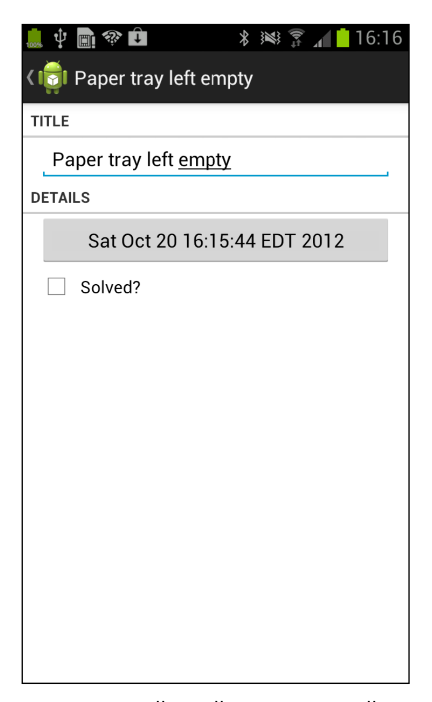


Рис. 16.9. Панель действий с включенной кнопкой Up

В методе `CrimeFragment.onCreateView(…)` вызовите `setDisplayHomeAsUpEnabled(true)`.

Листинг 16.8. Включение кнопки Up (CrimeFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
    }

    ...
}
```

Обратите внимание: задание этого свойства не активизирует выполнение функций кнопки Up. Оно всего лишь позволяет значку действовать как кнопке и отображает стрелку. Подключение придется выполнять самостоятельно. Для приложений, рассчитанных на API уровней 11–13, значок по умолчанию активизируется, но для появления стрелки все равно необходимо вызвать `setDisplayHomeAsUpEnabled(true)`.

В листинге 16.8 onCreateView(…) снабжается аннотацией @TargetApi. В принципе можно ограничиться аннотацией только setDisplayHomeAsUpEnabled(true), но метод onCreateView(…) скоро будет содержать большое количество кода, привязанного к конкретной версии API, поэтому мы вместо этого помечаем весь метод.

Запустите приложение CriminalIntent, перейдите на экран детализации и убедитесь в том, что рядом со значком приложения появляется треугольная стрелка.

Обработка кнопки Up

Обработка активного значка приложения производится так, как если бы он был существующей командой меню — переопределением onOptionsItemSelected(MenuItem). Следовательно, прежде всего следует сообщить FragmentManager, что CrimeFragment реализует обратные вызовы командного меню от имени активности.

Включите в метод CrimeFragment.onCreate(…) вызов setHasOptionsMenu(true), как это было сделано ранее для CrimeListFragment.

Листинг 16.9. Включение обработки меню (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    setHasOptionsMenu(true);
}
```

Вам не придется определять или заполнять команду значка приложения из файла XML. У нее имеется готовый идентификатор ресурса: android.R.id.home. В файле CrimeFragment.java переопределите метод onOptionsItemSelected(MenuItem), чтобы он реагировал на эту команду.

Листинг 16.10. Обработка на команду меню значка приложения (Home) (CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // Будет реализовано позднее
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Кнопка должна возвращать пользователя к списку. Конечно, можно создать интент и запустить экземпляр CrimePagerActivity:

```
Intent intent = new Intent(getActivity(), CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

Флаг FLAG_ACTIVITY_CLEAR_TOP приказывает Android провести поиск существующего экземпляра активности в стеке, и если он будет найден — вывести из стека все остальные активности, чтобы запускаемая активность была верхней.



Рис. 16.10. FLAG_ACTIVITY_CLEAR_TOP в действии

Однако существует другой, более правильный способ реализации иерархической навигации, основанный на использовании вспомогательного класса `NavUtils` и включении метаданных в манифест.

Начнем с метаданных. Откройте файл AndroidManifest.xml. Добавьте в объявление `CrimePagerActivity` следующий атрибут, назначающий `CrimeListActivity` его родителем.

Листинг 16.11. Добавление метаданных родительской активности (AndroidManifest.xml)

```
<activity android:name=".CrimePagerActivity"
  android:label="@string/app_name">
  <meta-data android:name="android.support.PARENT_ACTIVITY"
    android:value=".CrimeListActivity"/>
</activity>
...
```

Тег метаданных — своего рода «листок для заметок», приклеенный к активности. Такие заметки хранятся в системном объекте `PackageManager`, и любой желающий может получить значение из «заметки», если он знает ее имя. Вы можете создавать собственные пары «имя-значение» и читать их данные по мере необходимости. Эту конкретную пару определяет класс `NavUtils`, чтобы он мог узнать родителя заданной активности. Она особенно полезна в сочетании со следующим методом класса `NavUtils`:

```
public static void navigateUpFromSameTask(Activity sourceActivity)
```

В методе `CrimeFragment.onOptionsItemSelected(…)` сначала проверьте, существует ли родительская активность, обозначенная в метаданных, при помощи вызова `NavUtils.getParentActivityName(Activity)`. Если она существует, вызовите `navigateUpFromSameTask(Activity)` для перехода к родительской активности.

Листинг 16.12. Использование NavUtils (CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            if (NavUtils.getParentActivityName(getActivity()) != null) {
```

```
                NavUtils.navigateUpFromSameTask(getActivity());
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Если информация о родителе в метаданных отсутствует, отображать стрелку не нужно. Вернитесь к onCreateView(…) и проверьте наличие родителя перед вызовом setDisplayHomeAsUpEnabled(true).

Листинг 16.13. Нет родителя — нет стрелки (CrimeFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (NavUtils.getParentActivityName(getActivity()) != null) {
            getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }

    ...
}
```

Почему использование NavUtils лучше самостоятельного запуска активности? Во-первых, код NavUtils компактен и понятен. Кроме того, использование NavUtils обеспечивает централизацию отношений между активностями в манифесте. Если эти отношения изменятся, достаточно изменить строку в манифесте вместо того, чтобы возиться с кодом Java.

Другое преимущество заключается в том, что иерархические отношения отделяются от кода фрагмента. CrimeFragment может использоваться в разных активностях, которые могут иметь разных родителей; CrimeFragment все равно будет работать правильно.

Запустите приложение CriminalIntent. Создайте преступление и нажмите на значке приложения, чтобы вернуться к списку преступлений. Может, из жалкой двухуровневой иерархии CriminalIntent это и не очевидно, но метод navigateUpFromSameTask(Activity) реализует функциональность «перехода наверх» и поднимает пользователя на один уровень к родителю CrimePagerActivity.

Альтернативная команда меню

В этом разделе все, что мы узнали о меню, совместимости и альтернативных ресурсах, будет использовано для добавления команды меню, скрывающей и отображающей подзаголовок в панели действий CrimeListActivity.

Создание альтернативного файла меню

Команда меню, применяемая к панели действий, не должна быть видимой пользователям без панели действий. Следовательно, первым шагом должно быть создание альтернативного ресурса меню. Создайте в каталоге res проекта папку menu-v11. Скопируйте и вставьте файл fragment_crime_list.xml в эту папку.

В файле res/menu-v11/fragment_crime_list.xml добавьте команду меню Show Subtitle, которая будет отображаться на панели действий при наличии свободного места.

Листинг 16.14. Добавление команды меню Show Subtitle (res/menu-v11/fragment_crime_list.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        android:showAsAction="ifRoom|withText"/>
    <item android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        android:showAsAction="ifRoom"/>
</menu>
```

Добавьте в метод onOptionsItemSelected(…) обработку команды меню — включение подзаголовка на панели действия.

Листинг 16.15. Обработка команды меню Show Subtitle (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Обратите внимание: мы только отменяем предупреждения Android Lint, а не заключаем код панели действий в проверочную конструкцию. Проверка не нужна — этот код не может вызываться на старых устройствах, потому что команда меню R.id.menu_item_show_subtitle на них отображаться не будет.

Запустите приложение CriminalIntent на новом устройстве и включите подзаголовок. Затем запустите его на устройстве Froyo или Gingerbread (физическом или виртуальном). Нажмите кнопку меню и убедитесь в том, что команда Show Subtitle не отображается. Добавьте новое преступление и убедитесь в том, что приложение работает так же, как прежде.

Переключение текста команды

Теперь подзаголовок отображается, но текст команды меню остался неизменным: Show Subtitle. Было бы лучше, если бы текст команды и функциональность команды меню изменялись в зависимости от текущего состояния подзаголовка.

В методе onOptionsItemSelected(…) проверьте наличие подзаголовка при выборе команды меню и выполните соответствующие действия.

Листинг 16.16. Обработка в зависимости от наличия подзаголовка (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                item.setTitle(R.string.hide_subtitle);
            } else {
                getActivity().getActionBar().setSubtitle(null);
                item.setTitle(R.string.show_subtitle);
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Если панель действий не содержит подзаголовка, мы включаем подзаголовок и заменяем текст команды меню на Hide Subtitle. Если подзаголовок уже отображается, то он отключается, а команде меню возвращается текст Show Subtitle.

Запустите приложение CriminalIntent и убедитесь в том, что подзаголовок успешно скрывается и отображается.

«Да, и еще одно...»

Программирование Android часто напоминает беседы с детективом Коломбо из сериала. Вы уже думаете, что все прошло как по маслу и у следствия нет претензий. Но Android всегда поворачивается у двери и говорит: «Да, и еще одно...»

В нашем случае это повороты. Если отобразить подзаголовок, а потом повернуть устройство, подзаголовок исчезнет при создании интерфейса «с нуля». Для решения этой проблемы нужно определить поле для признака видимости подзаголовка, а также сохранить CrimeListFragment, чтобы значение переменной не терялось при поворотах.

В файле CrimeListFragment.java добавьте логическую переменную, затем в методе onCreate(…) сохраните CrimeListFragment и инициализируйте переменную.

Листинг 16.17. Инициализация переменных и сохранение CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {    private ArrayList<Crime>
mCrimes;
    private boolean mSubtitleVisible;
    private final String TAG = "CrimeListFragment";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        setRetainInstance(true);
        mSubtitleVisible = false;
    }
```

Затем в методе `onOptionsItemSelected(…)` задайте эту переменную при обработке выбора команды меню.

Листинг 16.18. Присваивание переменной subtitleVisible при обработке команды меню (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                mSubtitleVisible = true;
                item.setTitle(R.string.hide_subtitle);
            }
            else {
                getActivity().getActionBar().setSubtitle(null);
                mSubtitleVisible = false;
                item.setTitle(R.string.show_subtitle);
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Осталось проверить, может ли отображаться подзаголовок. В файле CrimeListFragment.java переопределите метод `onCreateView(…)` и назначьте подзаголовок, если переменная `mSubtitleVisible` содержит `true`.

Листинг 16.19. Подзаголовок назначается, если поле mSubtitleVisible истинно (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
```

```
        Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (mSubtitleVisible) {
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
        }
    }

    return v;
}
```

Также необходимо проверить состояние подзаголовка в `onCreateOptionsMenu(…)` и убедиться в том, что отображается правильный текст команды меню.

Листинг 16.20. Назначение текста команды меню с истинным значением mSubtitleVisible (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
    MenuItem showSubtitle = menu.findItem(R.id.menu_item_show_subtitle);
    if (mSubtitleVisible && showSubtitle != null) {
        showSubtitle.setTitle(R.string.hide_subtitle);
    }
}
```

Запустите приложение CriminalIntent. Отобразите подзаголовок, поверните устройство. Подзаголовок должен появиться в воссозданном представлении, как и ожидалось.

Упражнение. Пустое представление для списка

В настоящее время при запуске CriminalIntent отображает пустой список — большую черную пустоту. Мы должны предоставить пользователям что-то для взаимодействия при отсутствии элементов в списке.

Так как класс `ListView` является субклассом `AdapterView`, он поддерживает специальный вариант `View`, называемый «пустым представлением» и существующий именно для таких ситуаций. Если задать пустое представление, `ListView` будет автоматически переключаться между этим представлением при отсутствии элементов списка и отображением списка, если в нем есть хотя бы один элемент.

Для изменения пустого представления в коде используется следующий метод `AdapterView`:

```
public void setEmptyView(View emptyView)
```

Также можно создать в разметке XML макет, в котором задается как `ListView`, так и пустое представление. Если назначить им идентификаторы ресурсов `@android:id/list` и `@android:id/empty` соответственно, будет использоваться функция автоматического переключения.

Текущая реализация CrimeListFragment не заполняет свой макет в onCreateView(…), но для реализации пустого представления в макете это необходимо. Создайте для CrimeListFragment XML-ресурс макета, использующий FrameLayout в качестве корневого контейнера, с представлением ListView и другим представлением View, которое будет использоваться для пустого списка.

Пусть в пустом представлении выводится сообщение (например, «Список пуст»). Добавьте в представление кнопку, которая будет инициировать создание нового преступления, чтобы пользователю не пришлось обращаться к командному меню или панели действий.

17 Сохранение и загрузка локальных файлов

Почти каждому приложению необходимо место для хранения данных. В этой главе мы научим приложение CriminalIntent сохранять и загружать данные из файла JSON, хранящегося в файловой системе устройства.

Каждое приложение на устройстве Android имеет каталог в своей *песочнице* (sandbox). Хранение файлов в песочнице защищает их от других приложений и даже любопытных глаз пользователей (если только устройство не было «взломано» — в этом случае пользователь сможет делать все, что ему заблагорассудится).

Песочница каждого приложения представляет собой подкаталог каталога /data/data, имя которого соответствует имени пакета приложения. Для CriminalIntent полный путь к каталогу песочницы имеет вид /data/data/com.bignerdranch.android.criminalintent.

Знать путь к песочнице полезно, но запоминать его не нужно; если понадобится, вы всегда сможете получить его при помощи вспомогательных методов API.

Кроме песочницы, ваше приложение может хранить файлы во внешнем хранилище. Как правило, это SD-карта, которая может быть доступна (или не доступна) на устройство. На SD-карте могут храниться файлы и даже целые приложения, однако при этом приходится учитывать ряд факторов из области безопасности и программирования. Самый важный момент заключается в том, что доступ к файлам на внешнем хранилище не ограничивается вашим приложением — любой желающий может читать, записывать и удалять их. В этой главе основное внимание будет уделяться внутреннему (закрытому) хранилищу, но тот же API при необходимости может использоваться для работы с файлами внешнего хранилища. (Собственно, упражнение в конце этой главы посвящено именно этой теме.)

Сохранение и загрузка данных в CriminalIntent

Поддержка долгосрочного хранения данных в приложении включает два процесса: сохранение данных в файловой системе и их загрузка при запуске приложения.

Каждый процесс состоит из двух фаз. При сохранении данные сначала преобразуются в формат хранения, после чего результат записывается в файл. При загрузке все происходит наоборот: отформатированные данные сначала читаются из файла, а затем разбираются в формат, с которым работает приложение.

Для CriminalIntent форматом хранения будет формат JSON, а операции чтения и записи файлов осуществляются методами ввода-вывода класса Android `Context`. На рис. 17.1 представлена общая схема реализации сохранения и загрузки данных в CriminalIntent.

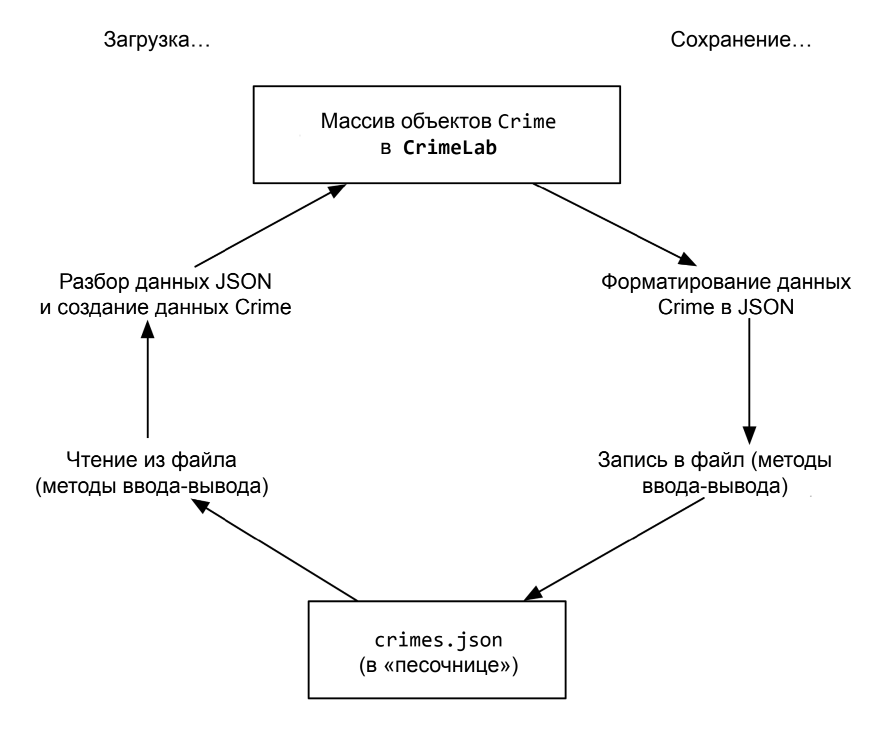


Рис. 17.1. Сохранение и загрузка в CriminalIntent

Формат *JSON* (JavaScript Object Notation) стал популярным в последнее время, особенно в области веб-служб. Android включает стандартный пакет `org.json`, классы которого предоставляют средства для создания и разбора файлов в формате JSON. В документации разработчика Android приведено описание `org.json`, а более подробную информацию о формате JSON можно получить по адресу *http://json.org*.

(XML — другой способ форматирования данных для записи в файл. Android также предоставляет классы и методы для работы с XML. Примеры их использования при разборе XML приведены в главе 26.)

Для работы с файлами, доступными для вашего приложения, удобнее всего использовать методы ввода-вывода, предоставляемые классом `Context` (суперклассом всех ключевых компонентов приложений: `Application`, `Activity` и `Service`, а также

нескольких других). Эти методы возвращают экземпляры стандартных классов Java — таких, как java.io.File или java.io.FileInputStream.

Сохранение преступлений в файле JSON

В приложении CriminalIntent класс CrimeLab будет отвечать за инициирование сохранения и загрузки данных, а механика создания и разбора объектов модели в формате JSON будет делегирована новому классу CriminalIntentJSONSerializer и существующему классу Crime.

Создание класса CriminalIntentJSONSerializer

Обязанности по записи существующего списка ArrayList объектов Crime в формат JSON делегированы классу CriminalIntentJSONSerializer.

Создайте этот класс в пакете com.bignerdranch.android.criminalintent. Оставьте его суперклассом java.lang.Object.

Затем добавьте код, приведенный в листинге 17.1. Не забудьте добавить команды import для классов, необходимых для работы кода, командой Eclipse Organize Imports.

Листинг 17.1. Реализация CriminalIntentJSONSerializer

```
public class CriminalIntentJSONSerializer {
    private Context mContext;
    private String mFilename;

    public CriminalIntentJSONSerializer(Context c, String f) {
        mContext = c;
        mFilename = f;
    }

    public void saveCrimes(ArrayList<Crime> crimes)
            throws JSONException, IOException {
        // Построение массива в JSON
        JSONArray array = new JSONArray();
        for (Crime c : crimes)
            array.put(c.toJSON());

        // Запись файла на диск
        Writer writer = null;
        try {
            OutputStream out = mContext
                .openFileOutput(mFilename, Context.MODE_PRIVATE);
            writer = new OutputStreamWriter(out);
            writer.write(array.toString());
        } finally {
            if (writer != null)
                writer.close();
        }
    }
}
```

Хотя код сериализации можно включить непосредственно в класс CrimeLab, выделение логики сериализации данных в формат JSON в автономный модуль имеет

свои преимущества. Модульное тестирование (unit-testing) класса упрощается, поскольку оно не зависит от функционирования других компонентов приложения. Кроме того, обратите внимание на то, что конструктор класса получает экземпляр `Context` и имя файла. Это означает, что класс может использоваться в разных местах без изменений, поскольку ему подходит любая реализация `Context`.

В методе `saveCrimes(ArrayList<Crime>)` мы создаем объект `JSONArray`, после чего вызываем метод `toJSON()` для каждого преступления в списке и добавляем результат в `JSONArray`. (В настоящее время вызов `toJSON()` вызывает ошибку, потому что метод еще не реализован в `Crime`. Сейчас мы займемся этим.)

Чтобы открыть файл и записать в него данные, используйте метод `Context.openFileOutput(…)`. При вызове передается имя файла и режим. Метод автоматически присоединяет имя файла к пути песочницы вашего приложения, создает файл и открывает его для записи. Если вы предпочитаете создавать и открывать файлы вручную, используйте вызов `Context.getFilesDir()`, но метод `openFileOutput(…)` необходим для создания файлов с разными разрешениями.

После того как файл будет открыт, запись в него осуществляется через стандартные интерфейсы Java. Классы `Writer`, `OutputStream` и `OutputStreamWriter` находятся в пакете `java.io`. Объект `OutputStream`, возвращаемый `openFileOutput(…)`, используется для создания нового объекта `OutputStreamWriter`, который обеспечивает преобразование между передаваемыми объектами Java `String` и низкоуровневым потоком байтов, которые `OutputStream` в конечном итоге записывает в файл.

Поддержка сериализации JSON в классе Crime

Чтобы сохранить массив `mCrimes` в формате JSON, необходимо иметь возможность сохранять отдельные экземпляры `Crime` в формате JSON. Добавьте в файл Crime.java следующие константы и реализацию `toJSON()`, которая сохраняет `Crime` в формате JSON и возвращает экземпляр класса `JSONObject` для включения в `JSONArray`.

Листинг 17.2. Реализация сериализации JSON (Crime.java)

```
public class Crime {

    private static final String JSON_ID = "id";
    private static final String JSON_TITLE = "title";
    private static final String JSON_SOLVED = "solved";
    private static final String JSON_DATE = "date";

    private UUID mId;
    private String mTitle;
    private boolean mSolved;
    private Date mDate = new Date();

    public Crime() {
        mId = UUID.randomUUID();
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_ID, mId.toString());
```

```
        json.put(JSON_TITLE, mTitle);
        json.put(JSON_SOLVED, mSolved);
        json.put(JSON_DATE, mDate.getTime());
        return json;
    }

    @Override
    public String toString() {
        return mTitle;
    }
```

Код использует методы класса `JSONObject` для преобразования данных `Crime` в формат, который может быть записан в файл как разметка JSON.

Сохранение объектов Crime в CrimeLab

После добавления `CriminalIntentJSONSerializer` и внесения изменений в `Crime` мы можем записать список преступлений в формате JSON и сохранить данные в файловой системе устройства.

Когда следует сохранять данные? Практическое правило для мобильных приложений гласит: сохраняйте данные как можно чаще, желательно сразу же после внесения изменений пользователем. Наш код уже передает изменения в данных в класс `CrimeLab`, так что будет логично сохранить при этом файл.

Слишком частое сохранение данных может замедлить работу приложения. Каждый раз, когда с данными что-то происходит, наш код перезаписывает весь файл преступлений. Приложение CriminalIntent невелико, так что запись не займет много времени. Для более основательных объемов желательно сохранять отдельные блоки данных без сохранения всех остальных — например, в базе данных SQLite. Использование SQLite в приложениях Android рассматривается в главе 34.

В файле CrimeLab.java создайте экземпляр `CriminalIntentJSONSerializer` в `onCreate(…)`. Затем определите метод `saveCrimes()`, который пытается сериализовать список и возвращает логический признак успеха. Добавьте команды регистрации, чтобы легко проверить результат операции.

Листинг 17.3. Сохранение изменений в CrimeLab (CrimeLab.java)

```
public class CrimeLab {
    private static final String TAG = "CrimeLab";
    private static final String FILENAME = "crimes.json";

    private ArrayList<Crime> mCrimes;
    private CriminalIntentJSONSerializer mSerializer;

    private static CrimeLab sCrimeLab;
    private Context mAppContext;

    private CrimeLab(Context appContext) {
        mAppContext = appContext;
        mCrimes = new ArrayList<Crime>();
    }
    ...
```

продолжение ⇨

Листинг 17.3 (*продолжение*)

```
    public void addCrime(Crime c) {
        mCrimes.add(c);
    }

    public boolean saveCrimes() {
        try {
            mSerializer.saveCrimes(mCrimes);
            Log.d(TAG, "crimes saved to file");
            return true;
        } catch (Exception e) {
            Log.e(TAG, "Error saving crimes: ", e);
            return false;
        }
    }
}
```

В нашем примере информация об ошибках для простоты регистрируется в журнале. В реальном приложении желательно привлечь внимание пользователя к неудачной попытке сохранения — например, при помощи `Toast` или диалогового окна.

Сохранение данных приложения в onPause()

Где следует вызывать `saveCrimes()`? Самый безопасный вариант — метод жизненного цикла `onPause()`. Если дожидаться `onStop()` или `onDestroy()`, можно упустить возможность сохранения. Приостановленная активность будет уничтожена, если ОС потребуется освободить память, а в этих обстоятельствах вызов `onStop()` или `onDestroy()` не гарантирован.

Листинг 17.4. Сохранение преступлений в файловой системе в методе onPause() (CrimeFragment.java)

```
    ...

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                NavUtils.navigateUpFromSameTask(getActivity());
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        CrimeLab.get(getActivity()).saveCrimes();
    }
}
```

Запустите приложение CriminalIntent. Добавьте пару преступлений и нажмите кнопку Home на устройстве; активность приостанавливается, а список преступлений сохраняется в файловой системе. Проверьте результат при помощи LogCat.

Загрузка данных из файловой системы

Теперь пройдемся в обратном направлении и заставим CriminalIntent загружать преступления из файловой системы при запуске приложения. Сначала добавьте в Crime.java конструктор, получающий `JSONObject`.

Листинг 17.5. Реализация Crime(JSONObject) (Crime.java)

```
public class Crime {
   ...
    public Crime() {
        mId = UUID.randomUUID();
    }

    public Crime(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        mTitle = json.getString(JSON_TITLE);
        mSolved = json.getBoolean(JSON_SOLVED);
        mDate = new Date(json.getLong(JSON_DATE));
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_ID, mId.toString());
        json.put(JSON_TITLE, mTitle);
        json.put(JSON_SOLVED, mSolved);
        json.put(JSON_DATE, mDate.getTime());
        return json;
    }
```

Затем в файле CriminalIntentJSONSerializer.java добавьте в класс `CriminalIntentJSONSerializer` метод загрузки данных из файловой системы.

Листинг 17.6. Реализация loadCrimes() (CriminalIntentJSONSerializer.java)

```
public CriminalIntentJSONSerializer(Context c, String f) {
    mContext = c;
    mFilename = f;
}

public ArrayList<Crime> loadCrimes() throws IOException, JSONException {
    ArrayList<Crime> crimes = new ArrayList<Crime>();
    BufferedReader reader = null;
    try {
        // Открытие и чтение файла в StringBuilder
        InputStream in = mContext.openFileInput(mFilename);
        reader = new BufferedReader(new InputStreamReader(in));
        StringBuilder jsonString = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            // Line breaks are omitted and irrelevant
            jsonString.append(line);
        }
        // Разбор JSON с использованием JSONTokener
        JSONArray array = (JSONArray) new JSONTokener(jsonString.toString())
            .nextValue();
```

продолжение ⇨

Листинг 17.6 (*продолжение*)

```
        // Построение массива объектов Crime по данным JSONObject
        for (int i = 0; i < array.length(); i++) {
            crimes.add(new Crime(array.getJSONObject(i)));
        }
    } catch (FileNotFoundException e) {
        // Происходит при начале "с нуля"; не обращайте внимания
    } finally {
        if (reader != null)
            reader.close();
    }
    return crimes;
}

public void saveCrimes(ArrayList<Crime> crimes) throws JSONException, IOException {
    // Построение массива в JSON
    JSONArray array = new JSONArray();
    for (Crime c : crimes)
        array.put(c.toJSON());
    ...
```

Мы используем комбинацию Java и классов JSON в сочетании с методом `openFileInput(…)` класса `Context` для чтения данных из файловой системы в строку и последующего разбора этой строки, содержащей объекты `JSONObject`, в `JSONArray`, а затем в объект `ArrayList`, который в конечном счете возвращается методом.

Обратите внимание на вызов `reader.close()` в блоке `finally`. Он гарантирует, что используемый файловый дескриптор будет освобожден даже при возникновении ошибки.

Наконец, в конструкторе `CrimeLab` данные преступлений должны быть загружены в `ArrayList` при первом обращении к синглетному объекту (вместо того, чтобы каждый раз создавать новый пустой объект). В файле CrimeLab.java внесите следующие изменения.

Листинг 17.7. Загрузка данных (CrimeLab.java)

```
public CrimeLab(Context appContext) {
    mAppContext = appContext;
    mSerializer = new CriminalIntentJSONSerializer(mAppContext, FILENAME);
    mCrimes = new ArrayList<Crime>();

    try {
        mCrimes = mSerializer.loadCrimes();
    } catch (Exception e) {
        mCrimes = new ArrayList<Crime>();
        Log.e(TAG, "Error loading crimes: ", e);
    }
}

public static CrimeLab get(Context c) {
    ...
}
...
```

Мы пытаемся загрузить существующие данные; если их нет, создается новый пустой список.

Теперь приложение CriminalIntent может сохранять данные между запусками. Новую функциональность можно протестировать разными способами. Запустите приложение, добавьте преступления и внесите изменения. Перейдите в другое приложение (например, браузер); вероятно, в этот момент ОС завершит CriminalIntent. Запустите приложение снова и убедитесь в том, что изменения сохранены. Также можно остановить и перезапустить приложение из Eclipse.

Теперь вы можете смело вводить всю подборку ужасных преступлений, совершаемых в вашем офисе. Данные сохраняются между запусками, вы можете использовать их при добавлении новых описаний и вам не придется заново вводить данные при каждом запуске приложения.

Упражнение. Использование внешнего хранилища

Файлы во внутреннем хранилище хорошо подходят для большинства приложений, особенно если вы беспокоитесь о конфиденциальности данных. Тем не менее некоторые приложения могут записывать данные во внешнее хранилище устройства, если оно доступно.

Например, внешнее хранилище удобно для передачи данных (музыки, фотографий, загруженной из Интернета информации) другим приложениям или пользователям. Кроме того, внешнее хранилище почти всегда обладает большим объемом, и в нем обычно хранятся большие файлы (например, видео).

Для записи во внешнее хранилище необходимо сделать две вещи. Сначала убедитесь в том, что внешнее хранилище доступно. Класс `android.os.Environment` содержит несколько полезных методов и констант, которые помогут вам в этом. Затем получите дескриптор каталога внешних файлов (найдите метод класса `Context`, который предоставит вам эту информацию). Все остальное делается практически так же, как в `CriminalIntentJSONSerializer`.

Для любознательных: файловая система Android и средства ввода-вывода Java

На уровне ОС Android работает в ядре Linux. В результате файловая система Android сходна с другими файловыми системами Linux и Unix. Имена каталогов разделяются символом косой черты (`/`), имена файлов могут содержать разнообразные символы, а регистр символов учитывается. Ваше приложение работает под конкретным идентификатор пользователя, используя преимущества модели безопасности Linux.

Android определяет местонахождение приложения в файловой системе по имени пакета Java. Сами приложения для распространения и установки упаковываются в файл APK, который хранится в каталоге /data/app с именем вида com.bignerdranch.android.criminalintent-1.apk. Обычно разработчику ничего не нужно знать об этом

каталоге, и на типичном устройстве обращение к нему напрямую возможно только после взлома устройства.

Обращение к файлам и каталогам

Для работы с файлами и каталогами, доступными для вашего приложения, удобнее всего использовать методы класса `Context` — суперкласса всех ключевых компонентов приложений: `Application`, `Activity` и `Service`, а также ряда других. Он предоставляет своим субклассам простой доступ к файлам с использованием методов, перечисленных в табл. 17.1.

Таблица 17.1. Основные методы для работы с файлами и каталогами в Context

Метод	**Назначение**
File getFilesDir()	Возвращает дескриптор каталога для закрытых файлов приложения
FileInputStream openFileInput(String name)	Открывает существующий файл для ввода (относительно каталога файлов)
FileOutputStream openFileOutput(String name, int mode)	Открывает существующий файл для вывода, возможно с созданием (относительно каталога файлов)
File getDir(String name, int mode)	Получает (и возможно, создает) подкаталог в каталоге файлов
String[] fileList()	Получает список имен файлов в главном каталоге файлов (например, для использования с openFileInput(String))
File getCacheDir()	Возвращает дескриптор каталога, используемого для хранения кэш-файлов. Будьте внимательны, поддерживайте порядок в этом каталоге и старайтесь использовать как можно меньше пространства

Большинство этих методов возвращает экземпляры стандартных классов Java — таких, как `java.io.File` или `java.io.FileInputStream`. Многие существующие функции API, которые работают с этими классами, используются точно так же, как в любом другом приложении Java. Android также поддерживает классы `java.nio.*`.

18 Контекстные меню и режим контекстных действий

В этой главе мы предоставим пользователям возможность удалять элементы списка. Операция выполняется долгим нажатием на элементе. Удаление является контекстным действием: оно относится к конкретному представлению на экране (один элемент списка), а не ко всему экрану.

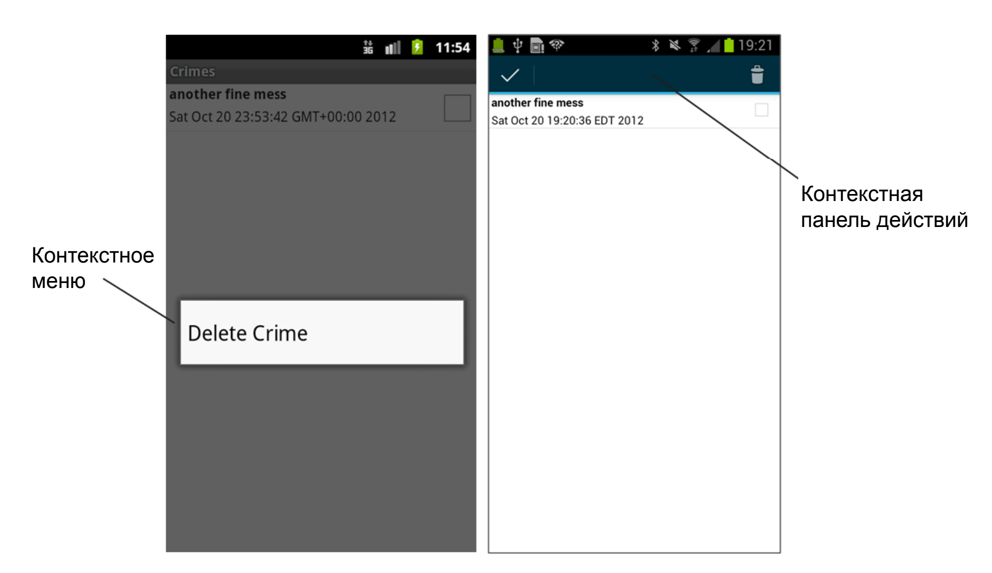


Рис. 18.1. Долгое нажатие на элементе списка

До версии Honeycomb контекстные действия представлялись контекстными меню. На более современных устройствах рекомендуется использовать контекстную панель действий. Контекстная панель действий размещается поверх панели действий активности и содержит информацию о доступных действиях.

С командными меню проблема разных уровней API решается просто: вы определяете один ресурс меню и реализуете один набор обратных вызовов. ОС каждого устройства определяет, как следует отображать элементы меню.

С контекстными действиями ситуация усложняется. Вы по-прежнему определяете один ресурс меню, но реализуете два разных набора обратных вызовов — для контекстной панели действий и для экранных контекстных меню.

В этой главе мы реализуем контекстное действие для устройств с API уровня 11 и выше и контекстное меню для устройств Froyo и Gingerbread.

(Возможно, вы видели старые устройства с контекстными панелями действий в приложения. Как правило, такие приложения используют стороннюю библиотеку ActionBarSherlock для дублирования функциональности панелей действий в более ранних API. Мы еще вернемся к ActionBarSherlock в конце этой главы.)

Определение ресурса контекстного меню

Создайте в каталоге res/menu/ новый файл XML с именем crime_list_item_context.xml, выберите для него корневой элемент menu. Затем добавьте один элемент item, представленный в листинге 18.1.

Листинг 18.1. Контекстное меню для списка преступлений (crime_list_item_context.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_crime"
      android:icon="@android:drawable/ic_menu_delete"
      android:title="@string/delete_crime" />
</menu>
```

Этот ресурс меню будет использоваться для обеих реализаций.

Реализация контекстного меню

Начнем с создания контекстного меню. Обратные вызовы Fragment аналогичны тем, которые использовались нами для командного меню в главе 16. Контекстное меню заполняется следующим методом:

```
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenu.ContextMenuInfo menuInfo)
```

Для обработки выбора команды контекстного меню реализуется следующий метод Fragment:

```
public boolean onContextItemSelected(MenuItem item)
```

Создание контекстного меню

В файле CrimeListFragment.java реализуйте метод onCreateContextMenu(…) для заполнения ресурса меню и построения контекстного меню на его основе.

Листинг 18.2. Создание контекстного меню (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        default:
            return super.onOptionsItemSelected(item);
    }
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
}
```

В отличие от onCreateOptionsMenu(…) этот метод обратного вызова не передает экземпляр MenuInflater, поэтому сначала следует получить экземпляр MenuInflater, связанный с CrimeListActivity. При вызове MenuInflater.inflate(…) передается идентификатор ресурса меню и экземпляр ContextMenu для заполнения экземпляра меню элементами, определенными в файле меню.

Пока в нашем приложении используется только один ресурс контекстного меню, поэтому этот ресурс заполняется независимо от того, на каком представлении было сделано долгое нажатие. Если приложение содержит несколько ресурсов контекстного меню, заполняемый ресурс определяется проверкой идентификатора объекта View, переданного onCreateContextMenu(…).

Регистрация контекстного меню

По умолчанию долгое нажатие на представлении не инициирует создание контекстного меню. Представление необходимо зарегистрировать для вызова контекстного меню при помощи следующего метода Fragment, с передачей соответствующего представления:

```
public void registerForContextMenu(View view)
```

В нашем случае контекстное меню должно появиться при щелчке на любом элементе списка. К счастью, регистрировать представления всех отдельных элементов списка не нужно. Вместо этого можно зарегистрировать экземпляр ListView, который автоматически регистрирует элементы списка.

В методе CrimeListFragment.onCreateView(…) получите ссылку на ListView и зарегистрируйте ее.

Листинг 18.3. Регистрация ListView для контекстного меню (CrimeListFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
```

продолжение ⇨

Листинг 18.3 (*продолжение*)

```
            Bundle savedInstanceState) {
        View v = super.onCreateView(inflater, parent, savedInstanceState);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            if (mSubtitleVisible) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
            }
        }

        ListView listView = (ListView)v.findViewById(android.R.id.list);
        registerForContextMenu(listView);

        return v;
    }
```

Идентификатор ресурса `android.R.id.list` используется для получения экземпляра `ListView`, находящегося под управлением `ListFragment`. Класс `ListFragment` также содержит метод `getListView()`, но в `onCreateView(…)` этот метод использоваться не может, потому что `getListView()` возвращает `null` до возвращения из `onCreateView(…)`.

Запустите приложение CriminalIntent, сделайте долгое нажатие на элементе списка и убедитесь в том, что на экране появляется контекстное меню с командой Delete Crime.

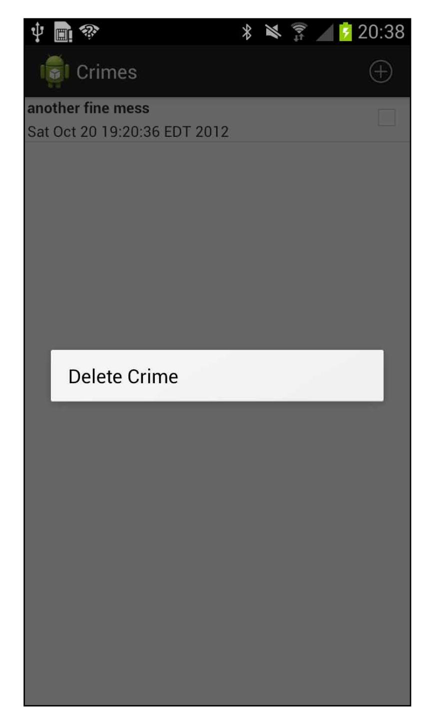


Рис. 18.2. Контекстное меню появляется при долгом нажатии

Реакция на действие

Для подключения команды меню нам понадобится метод, удаляющий объект преступления из модели. В файле CrimeLab.java добавьте метод `deleteCrime(Crime)`.

Листинг 18.4. Добавление метода для удаления преступления (CrimeLab.java)

```
public void addCrime(Crime c) {
    mCrimes.add(c);
}

public void deleteCrime(Crime c) {
    mCrimes.remove(c);
}
```

Следующий шаг — обработка выбора команды меню в методе `onContextItemSelected(MenuItem)`. У `MenuItem` имеется идентификатор, представляющий выбранную команду. Однако знать, что пользователь хочет удалить преступление из списка, недостаточно. Необходимо знать, *какое* преступление нужно удалить.

Эту информацию можно получить вызовом метода `getMenuInfo()` для `MenuItem`. Метод возвращает экземпляр класса, реализующего интерфейс `ContextMenu.ContextMenuInfo`.

В классе `CrimeListFragment` добавьте реализацию `onContextItemSelected(MenuItem)`, которая использует информацию меню и адаптер для определения того, на каком объекте `Crime` было сделано долгое нажатие. Далее этот объект `Crime` удаляется из модели.

Листинг 18.5. Прослушивание выбора команды контекстного меню (CrimeListFragment.java)

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)item.getMenuInfo();
    int position = info.position;
    CrimeAdapter adapter = (CrimeAdapter)getListAdapter();
    Crime crime = adapter.getItem(position);

    switch (item.getItemId()) {
        case R.id.menu_item_delete_crime:
            CrimeLab.get(getActivity()).deleteCrime(crime);
            adapter.notifyDataSetChanged();
            return true;
    }
    return super.onContextItemSelected(item);
}
```

В листинге 18.5 метод `getMenuInfo()` возвращает экземпляр `AdapterView.AdapterContextMenuInfo`, потому что `ListView` является субклассом `AdapterView`. Мы

преобразуем тип результата `getMenuInfo()` и получаем подробную информацию о выбранном элементе списка, включая его позицию в наборе данных. Позиция используется для получения искомого объекта `Crime`.

Запустите приложение CriminalIntent, добавьте новое преступление и удалите его долгим нажатием. (Чтобы имитировать долгое нажатие в эмуляторе, удерживайте нажатой левую кнопку мыши до появления меню.)

Реализация режима контекстных действий

Код, написанный нами для удаления преступления из контекстного меню, будет работать на любом устройстве Android. Например, на рис. 18.2 показано контекстное меню на устройстве с Jelly Bean.

Тем не менее на новых устройствах для отображения контекстных действий рекомендуется использовать долгое нажатие на представлении, выведенном на экран в *режиме контекстных действий*. Когда экран находится в режиме контекстных действий, элементы, определенные в контекстном меню, отображаются на контекстной панели действий, наложенной на обычную панель действий. Контекстные панели действий удобны тем, что они не загромождают экран.

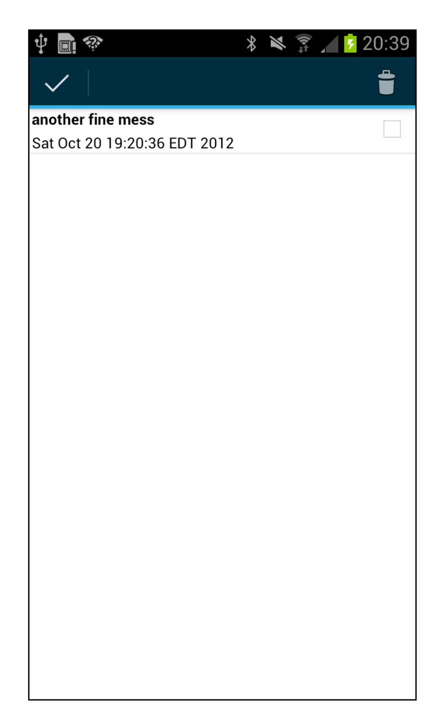


Рис. 18.3. Контекстная панель действий появляется при долгом нажатии

Чтобы реализовать контекстную панель действий, необходимо написать специальный код, отдельный от кода контекстного меню. Кроме того, код контекстной панели

действий содержит классы и методы, недоступные в Froyo и Gingerbread, поэтому необходимо проследить за тем, чтобы в таких ситуациях этот код не вызывался.

Множественное выделение

Когда представление списка находится в режиме контекстных действий, вы можете включить режим множественного выделения элементов списка. Любое действие, выбранное на контекстной панели действий, будет применяться ко всем выделенным представлениям.

В методе `CrimeListFragment.onCreateView(…)` назначьте представлению списка режим `CHOICE_MODE_MULTIPLE_MODAL`. Используйте константы версии сборки для отделения кода, регистрирующего `ListView`, от кода, назначающего режим выделения.

Листинг 18.6. Назначение режима выделения

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    ...
    ListView listView = (ListView)v.findViewById(android.R.id.list);

    if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
        // Контекстные меню для Froyo и Gingerbread
        registerForContextMenu(listView);
    } else {
        // Контекстная панель действий для Honeycomb и выше
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    }

    return v;
}
```

Методы обратного вызова режима действий в представлении списка

Следующий шаг — назначение для `ListView` слушателя, реализующего интерфейс `AbsListView.MultiChoiceModeListener`. Этот интерфейс содержит следующий метод, который вызывается при выделении или отмене выделения представления.

```
public abstract void onItemCheckedStateChanged(ActionMode mode, int position,
    long id, boolean checked)
```

Интерфейс `MultiChoiceModeListener` реализует другой интерфейс — `ActionMode.Callback`. Когда экран переходит в режим контекстных действий, создается экземпляр класса `ActionMode`, а методы `ActionMode.Callback` вызываются в разных точках жизненого цикла `ActionMode`. `ActionMode.Callback` содержит четыре обязательных метода:

```
public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
```

Вызывается при создании `ActionMode`. Именно здесь заполняется ресурс контекстного меню, которое будет отображаться на контекстной панели действий.

```
public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
```

Вызывается после `onCreateActionMode(…)` и каждый раз, когда существующую контекстную панель действий необходимо актуализировать новыми данными.

```
public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
```

Вызывается, когда пользователь выбирает действие. Здесь программируется обработка контекстных действий, определенных в ресурсе меню.

```
public abstract void onDestroyActionMode(ActionMode mode)
```

Вызывается перед уничтожением `ActionMode` из-за того, что пользователь отменил режим действий или выбранное действие было обработано. Реализация по умолчанию снимает выделение с представления(-й). Вы также можете обновить фрагмент любой необходимой информацией о том, что произошло в режиме контекстных действий.

В методе `CrimeListFragment.onCreateView(…)` назначьте слушателя, реализующего `MultiChoiceModeListener` для представления списка. В нашем случае что-то делать нужно только в методах `onCreateActionMode(…)` и `onActionItemClicked(ActionMode, MenuItem)`.

Листинг 18.7. Назначение слушателя MultiChoiceModeListener (CrimeListFragment.java)

```
...
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    public void onItemCheckedStateChanged(ActionMode mode, int position,
            long id, boolean checked) {
        // Метод является обязательным, но не используется
        // в этой реализации
    }

    // Методы ActionMode.Callback
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.crime_list_item_context, menu);
        return true;
    }

    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false;
        // Метод является обязательным, но не используется
        // в этой реализации
    }

    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_delete_crime:
                CrimeAdapter adapter = (CrimeAdapter)getListAdapter();
                CrimeLab crimeLab = CrimeLab.get(getActivity());
                for (int i = adapter.getCount() - 1; i >= 0; i--) {
```

```
                        if (getListView().isItemChecked(i)) {
                            crimeLab.deleteCrime(adapter.getItem(i));
                        }
                    }
                    mode.finish();
                    adapter.notifyDataSetChanged();
                    return true;
                default:
                    return false;
            }
        }
        public void onDestroyActionMode(ActionMode mode) {
            // Метод является обязательным, но не используется
            // в этой реализации
        }
    });
    return v;
}
```

Если вы использовали функцию автозавершения Eclipse для создания этого интерфейса, обратите внимание: заглушка, сгенерированная для `onCreateActionMode(…)`, возвращает `false`. Обязательно измените возвращаемое значение на `true`; возвращение `false` отменяет создание режима действий.

В коде метода `onCreateActionMode(…)` обратите внимание на то, что мы получаем экземпляр `MenuInflater` от объекта `ActionMode`, а не от активности. Режим действий содержит подробную информацию для настройки контекстной панели действий. Например, вызов `ActionMode.setTitle(…)` позволяет назначить контекстной панели действий специальный заголовок. Объект `MenuInflater` активности не будет знать об этом заголовке.

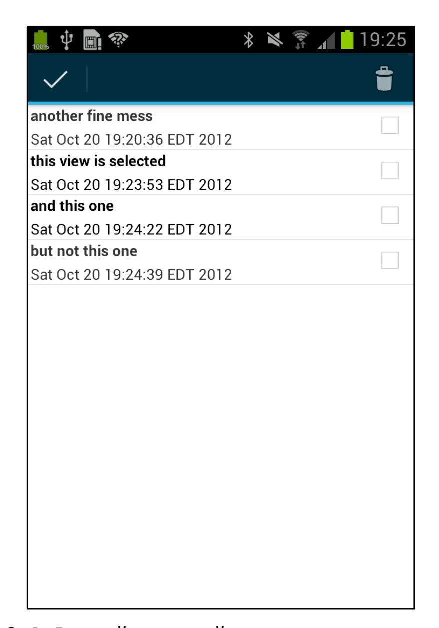


Рис. 18.4. Второй и третий элемент списка выделены

В методе onActionItemClicked(…) мы удаляем один или несколько объектов Crime из CrimeLab, а затем перезагружаем список для отражения внесенных изменений. Вызов ActionMode.finish() подготавливает объект режима действий к уничтожению.

Запустите приложение CriminalIntent. Сделайте долгое нажатие на элементе списка, чтобы выделить его и войти в режим контекстных действий. Выберите другие элементы списка (обычным) нажатием. Снятие выделения с ошибочно выделенных элементов осуществляется так же. Удалите пару преступлений. Нажатие кнопки удаления завершает режим действий и возвращает приложение к обновленному списку преступлений.

Также можно отменить режим действий нажатием на значке у левого края панели контекстной панели действий. При этом режим действий завершается, а приложение возвращается к списку без внесения каких-либо изменений.

Хотя эта реализация работает, по внешнему виду списка трудно понять, какие элементы были выделены. Проблема решается изменением фона элементов списка при выделении.

Изменение фона выделенных элементов

Иногда представление должно изменяться в зависимости от своего текущего состояния. В нашем случае фон элемента списка должен изменяться при его переходе в *активное* состояние, чтобы привлечь внимание пользователя.

Изменение фона на основании состояния представления может осуществляться при помощи *графической метки списка состояний*. Такая метка представляет собой ресурс XML. В ресурсе указывается имя графической метки (растрового изображения или цвета) и перечисляется список состояний, в которых эта метка должна отображаться. (Другие состояния, в которых может находиться представление, перечислены в документации StateListDrawable.)

В отличие от других графических ресурсов, графические метки списка состояний не связываются с плотностью пикселов, а следовательно, размещаются в каталоге drawable без уточнений. Создайте каталог res/drawable и новый файл XML с именем res/drawable/background_activated.xml. Назначьте файлу корневой элемент selector и добавьте разметку, приведенную в листинге 18.8.

Листинг 18.8. Простая графическая метка списка состояний (res/drawable/background_activated.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
  <item
    android:state_activated="true"
    android:drawable="@android:color/darker_gray"
    />
</selector>
```

Эта разметка означает: «Когда представление, которое ссылается на эту графическую метку, находится в активном состоянии, использовать значение android:drawable. В противном случае ничего не делать». Если задать android:state_activated

значение `false`, то метка будет использоваться в тех случаях, когда представление *не находится* в активном состоянии.

Измените файл res/layout/list_item_crime.xml и включите в него ссылку на графическую метку.

Листинг 18.9. Изменение фона элемента списка (res/layout/list_item_crime.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@drawable/background_activated">
  ...
</RelativeLayout>
```

Снова запустите приложение CriminalIntent. На этот раз выделенные элементы будут сразу видны на общем фоне.

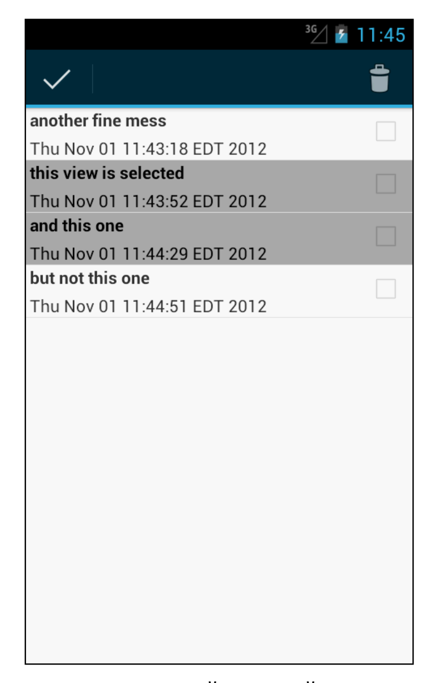


Рис. 18.5. Сразу видно, что второй и третий элемент списка выделены

О графических метках списков состояний более подробно рассказано в главе 25.

Реализация режима контекстных действий в других представлениях

Реализация контекстной панели действий, использованная в этой главе, хорошо подходит для `ListView`. Она также будет работать с `GridView` — субклассом

AdapterView, который мы будем использовать в главе 26. Но что, если вам понадобится контекстная панель действий для представления, которое не является ни ListView, ни GridView?

Прежде всего назначьте слушателя, реализующего View.OnLongClickListener. В реализации этого слушателя создайте экземпляр ActionMode вызовом Activity.startActionMode(…). (При использовании MultiChoiceModeListener экземпляр ActionMode создается за вас.)

Единственный параметр startActionMode(…) содержит объект, реализующий интерфейс ActionMode.Callback. Вы создаете реализацию ActionMode.Callback, включающую четыре метода жизненного цикла ActionMode, которые были реализованы ранее:

```
public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
public abstract void onDestroyActionMode(ActionMode mode)
```

Либо создайте объект, реализующий ActionMode.Callback, и передайте его startActionMode(…), либо передайте анонимную реализацию прямо в вызове startActionMode(…).

Совместимость: отход или дублирование?

В этой главе мы использовали стратегию совместимости, называемую *корректным отходом* (gracious fallback): приложение использует новые возможности и код на новых платформах и отходит к старой функциональности на старых устройствах. Для этого мы проверяем версию SDK во время выполнения.

Корректный отход — не единственная стратегия. Вместо отхода к старым платформам можно использовать идентичную функцию, имитирующую возможности новых платформ. Мы назовем эту стратегию *дублированием* (duplication).

Стратегии дублирования делятся на две группы: *дублирование по необходимости* и *заменяющее дублирование*. В первом случае замена производится только на старых платформах, на которых настоящая функциональность недоступна. Заменяющее дублирование используется постоянно — даже на платформах, на которых имеется собственная поддержка этой функциональности.

Использование библиотеки поддержки для работы с фрагментами — пример стратегии заменяющего дублирования. Ваши приложения всегда используют заменяющий класс android.support.v4.app.Fragment — даже при запуске на устройствах, на которых класс android.app.Fragment доступен.

В библиотеке поддержки нет дубликата для панели действий, но существует целый ряд сторонних реализаций панели действий, которые могут использоваться для обеспечения совместимости на старых устройствах. Если приложение с панелью действий работает на телефоне с Gingerbread, значит, оно использует одну из сторонних библиотек. Самая лучшая и популярная из них — библиотека ActionBarSherlock Джейка Уортона (Jake Wharton) — см. *http://www.actionbarsherlock.com*. Библиотека ActionBarSherlock базируется на новейших исходных кодах Android

и предоставляет поддержку дублирования по необходимости для функциональности панелей действий во всех предыдущих версиях Android. Дополнительная информация о библиотеке ActionBarSherlock и ее использовании приведена в разделе «Для любознательных» и втором упражнении в конце главы.

(Знающие люди говорят, что поддержка дубликатной панели действий появится и в библиотеке поддержки. Этот день для многих станет праздником.)

Какая стратегия лучше? У стратегий дублирования несомненные преимущества. Главное заключается в том, что поведение кода остается неизменным независимо от используемой версии Android. Это относится в первую очередь к стратегиям заменяющего дублирования, при которых во всех версиях Android выполняется совершенно одинаковый код. Проектировщиков дублирование тоже устраивает, потому что им приходится планировать только один вариант пользовательского интерфейса. К ним присоединяются тестировщики, которым достаточно протестировать только один набор взаимодействий. Наконец, при стратегии дублирования ваше приложение выглядит как новая разработка для ICS или Jelly Bean еще до того, как пользователь выполнит обновление.

Впрочем, у стратегии дублирования есть два больших недостатка. Во-первых, чтобы ваше приложение имело современный вид, приходится полагаться на сторонние библиотеки. Собственно, именно по этой причине в книге для панели действий использовалась стратегия корректного отхода. Во-вторых, такое приложение может казаться чужеродным на фоне остальных приложений устройства. Если в вашем приложении используется собственный вариант дизайна, это не создаст проблем, поскольку оно все равно будет отличаться от других. Но если вы намеревались соблюдать стиль стандартных приложений вашего телефона, такое приложение будет выделяться, как бельмо на глазу.

Упражнение. Удаление из CrimeFragment

Пользователям будет удобно иметь возможность удалять преступления как из списка, так и из представления детализации. В этом случае удаление будет применяться к экрану в целом, поэтому такое действие должно размещаться в командном меню или на панели действий. Реализуйте возможность удаления преступлений в CrimeFragment.

Для любознательных: ActionBarSherlock

Насколько бы важно ни было понимание принципов работы стандартных библиотек Android и их поведения на разных устройствах и версиях ОС, на практике разработчики часто забывают об этих проблемах. Безусловно, больше всего проблем с совместимостью создает панель действий, о которой было рассказано в этой главе и главе 16.

Библиотека ActionBarSherlock (или ABS, как ее чаще называют) предназначена исключительно для решения этих проблем. Она предоставляет реализацию панели действий для старых версий, а также набор классов, которые используют системную

версию в случае ее доступности или ее имитацию при необходимости. Библиотека доступна по адресу *http://www.actionbarsherlock.com*. Библиотека также предоставляет совместимые версии новых тем Android, включающих панель действий.

Если из этого описания вам показалось, что ABS представляет собой библиотеку поддержки с расширенной функциональностью, вы правы. Однако в отличие от библиотеки поддержки, ABS предоставляет темы и идентификаторы ресурсов. Это означает, что вместо простого jar-файла ABS распространяется в формате проекта библиотеки Android. Проект библиотеки напоминает обычный проект Android, но вместо независимого приложения он строит библиотеку, которая используется другими приложениями. Это позволяет Android включать в построение любые дополнительные ресурсы Android, предоставляемые библиотекой. Любая библиотека, предоставляющая дополнительные ресурсы Android, должна быть оформлена в виде проекта библиотеки, а не в виде jar-файла.

Поскольку ABS представляет собой проект библиотеки, для интеграции ABS в ваш проект необходимо:

- загрузить и распаковать исходный код;
- импортировать исходный код в проект Eclipse с именем ActionBarSherlock;
- добавить ссылку на новый проект ActionBarSherlock в CriminalIntent;
- обновить CriminalIntent для использования классов поддержки `ActionBarSherlock`.

(Если вы захотите выполнить эту короткую инструкцию по интеграции `ActionBarSherlock`, сначала создайте копию CriminalIntent. Наличие экземпляра CriminalIntent без кода ABS упростит чтение остальных глав).

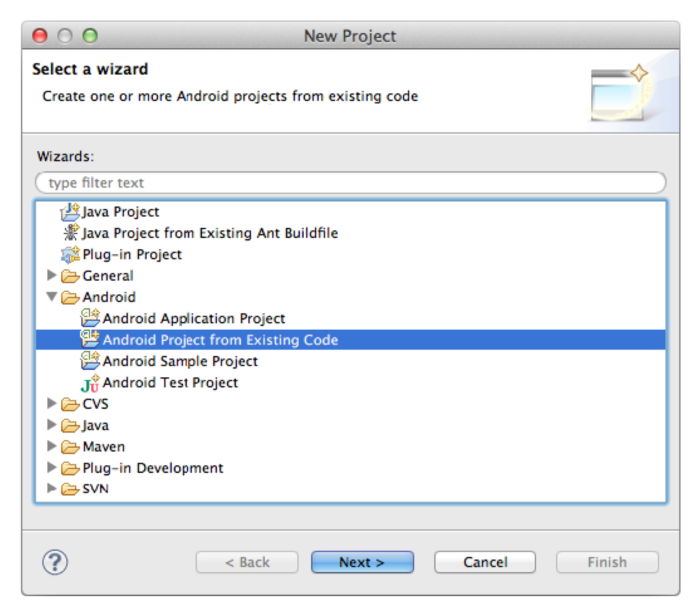


Рис. 18.6. Создание проекта Android на базе существующего кода

Чтобы загрузить ABS, откройте страницу *http://www.actionbarsherlock.com/download.html*, щелкните на ссылке загрузки архива в формате zip или tgz (выбирайте любой) и распакуйте загруженный архив.

Затем, чтобы создать новый проект в Eclipse, щелкните правой кнопкой мыши на панели Package Explorer и выберите команду New ▸ Project....

Вместо нового проекта Android нужно создать проект, включающий загруженный исходный код; выберите команду Android Project From Existing Code (рис. 18.6).

В этом диалоговом окне вам будет предложено выбрать корневой каталог для поиска существующего кода. Щелкните на кнопке Browse... и перейдите в каталог, в который была загружена библиотека ABS. При распаковке файла были созданы три вложенные папки: library, samples и website. Выберите library, щелкните на кнопке Open, затем на кнопке Finish.

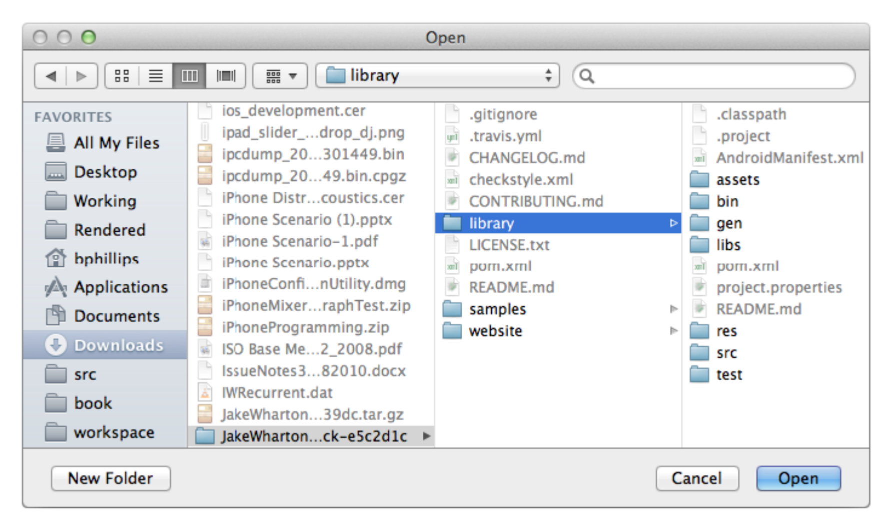


Рис. 18.7. Выбор папки с библиотекой ABS

При создании вашего проекта Eclipse присваивает ему имя library — не самое содержательное; щелкните правой кнопкой мыши, выберите команду Refactor ▸ Rename... и введите имя проекта ActionBarSherlock.

Остается сделать последний шаг — добавить ссылку на проект библиотеки в CriminalIntent. Щелкните правой кнопкой мыши на строке CriminalIntent на панели Package Explorer и выберите команду Properties... Выберите Android и взгляните на нижнюю часть окна, которая должна выглядеть примерно так.

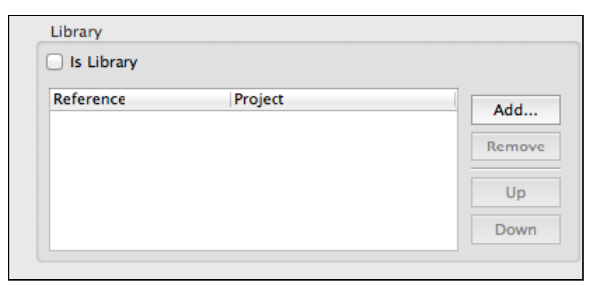


Рис. 18.8. Ссылки на библиотеки Android

В этом разделе перечисляются ссылки на проекты библиотек Android. Пока ни одной ссылки нет, а список пуст. Щелкните на кнопке Add...

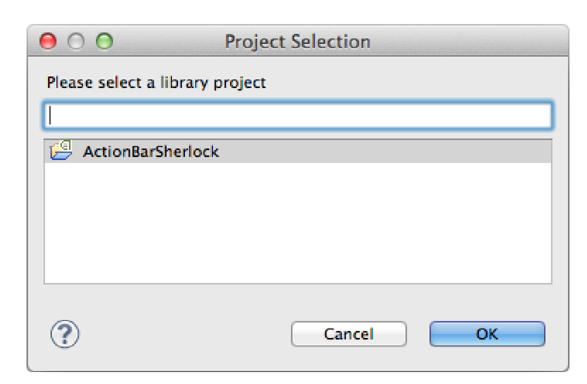


Рис. 18.9. Добавление ссылки на ActionBarSherlock

Наконец, сделайте двойной щелчок на строке ActionBarSherlock, чтобы добавить ссылку на проект библиотеки.

Упражнение. Использование ActionBarSherlock

Теперь, после добавления ссылки на проект `ActionBarSherlock`, все готово к его интеграции в CriminalIntent. Библиотека ABS работает по тому же принципу, что и библиотека поддержки — она предоставляет альтернативные версии таких основополагающих классов Android, как `Activity`, `Fragment` и `ActionBar`. Многие (но не все) имена классов ABS начинаются с префикса `Sherlock`, чтобы их было проще отличить от классов библиотеки поддержки.

Становится примерно понятно, что вам предстоит сделать. Классы фрагментов и активностей имеют префикс `Sherlock-`, но у классов меню этого префикса нет.

Базовая интеграция ABS в CriminalIntent

Последовательность действий по базовой интеграции ABS в проект:

- Измените объявления классов `SingleFragmentActivity` и `CrimePagerActivity` так, чтобы они были производными от `SherlockFragmentActivity` (вместо `FragmentActivity`).
- Измените все объявления классов фрагментов так, чтобы они были производными от `SherlockFragment`, `SherlockDialogFragment` или `SherlockListFragment` (вместо версий этих классов из библиотеки поддержки).
- Замените ссылки на `Menu`, `MenuItem` и `MenuInflater` ссылками на соответствующие реализации из `com.actionbarsherlock.view`.

Первые два шага просты — вы просто изменяете имена суперклассов. Когда это будет сделано, в `CrimeFragment` и `CrimeListFragment` появятся многочисленные

ошибки. Чтобы избавиться от них, необходимо выполнить третий шаг, который сложнее двух предыдущих.

Для CrimeFragment проблема решается просто: удалите относящиеся к меню директивы import в начале файла, проведите организацию импорта комбинацией клавиш Command+Shift+O/Ctrl+Shift+O и выберите версии com.actionbarsherlock.view.

Однако если вы попытаетесь проделать то же самое с CrimeListFragment, возникнут проблемы. Дело в том, что CrimeListFragment использует контекстные меню и MultiChoiceModeListener, для которых необходимы оригинальные версии классов меню из библиотеки Android.

Что делать? Вместо обходного решения с организацией импорта придется полностью уточнить ссылки на типы MenuItem, Menu и MenuInflater в onCreateOptionsMenu(…) и onOptionsItemSelected(…). Возьмем для примера следующий код:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}
```

Его необходимо записать в таком виде:

```
@Override
public void onCreateOptionsMenu(com.actionbarsherlock.view.Menu menu,
        com.actionbarsherlock.view.MenuInflater inflater) {
    ...
}
```

Интеграция более высокого уровня

Если вы выполнили первое упражнение, значит, вы интегрировали ABS в свой проект. Однако для полноценного использования библиотеки необходимо избавиться от кода совместимости. Для этого вызовите в коде getSherlockActivity().getSupportActionBar() вместо getActivity().getActionBar(). Панель действий SherlockActivity доступна всегда, поэтому ее использование позволит обойтись без кода защитной проверки. После этого можно переместить файл res/menu-v11/fragment_crime_list.xml в res/menu, чтобы избавиться от переключения версий ОС на уровне конфигурации.

Интеграция еще более высокого уровня

Ищете приключений? Следующим шагом на пути к идентичному поведению CriminalIntent в разных версиях является отказ от использования MultiChoiceModeListener и контекстных меню. Удалить код контекстного меню несложно, но для замены MultiChoiceModeListener потребуется имитация функциональности.

Как это сделать? Прежде всего можно использовать классический режим выбора ListView: ListView.CHOICE_MODE_MULTIPLE. В этой главе мы использовали режим ListView.CHOICE_MODE_MULTIPLE_MODAL, который работает только в новых версиях

Android. Режим `ListView.CHOICE_MODE_MULTIPLE` не поддерживает современное поведение модальных долгих нажатий, но у него есть свое преимущество: он работает во всех версиях Android. Выбор режима выделения `ListView.CHOICE_MODE_MULTIPLE` для `ListView` позволит выделять несколько элементов списка. Чтобы запретить выделение, верните значение `ListView.CHOICE_MODE_NONE`.

Следующим шагом должна стать имитация модального поведения панели действий, реализованного `CHOICE_MODE_MULTIPLE_MODAL`. Чтобы сделать это способом, универсальным для всех версий, следует вызвать `getSherlockActivity().startActionMode()`. Будьте внимательны и используйте версию, получающую `com.actionbarsherlock.view.ActionMode.Callback`, а не обычную версию Android.

Наконец, необходимо организовать обнаружение долгих нажатий. Для этого можно передать слушателя `OnItemLongClickListener` при вызове `ListView.setOnItemLongClickListener(…)`.

19 Камера I: Viewfinder

К описанию места преступления полезно приложить фотографию. В следующих двух главах мы добавим в приложение возможность создания снимков через API камеры.

API камеры при всей своей мощности не отличаются компактностью или простотой. Вам придется вводить большой объем кода, а время от времени — сражаться с достаточно сложными концепциями, что может показаться перебором. Пожалуй, кто-то подумает: «Мне нужно сделать простую фотографию. Неужели для этого нет стандартного интерфейса?»

Есть. С камерой можно взаимодействовать через неявный интент. На большинстве устройств Android установлено приложение для работы с камерой, которое прослушивает интент, созданный с кодом `MediaStore.ACTION_IMAGE_CAPTURE`. Вы научитесь работать с неявными интентами в главе 21.

К сожалению, на момент написания книги в интерфейсе неявных интентов присутствовала ошибка, из-за которой на многих устройствах сохранение полноразмерной фотографии становилось невозможным. Если приложению достаточно миниатюр, неявные интенты работают хорошо. Однако приложению CriminalIntent необходимы более крупные фотографии, поэтому вам придется осваивать API камеры.

В этой главе мы создадим активность на основе фрагмента, а также используем класс `SurfaceView` и камеру для вывода «живого» предварительного изображения с камеры.

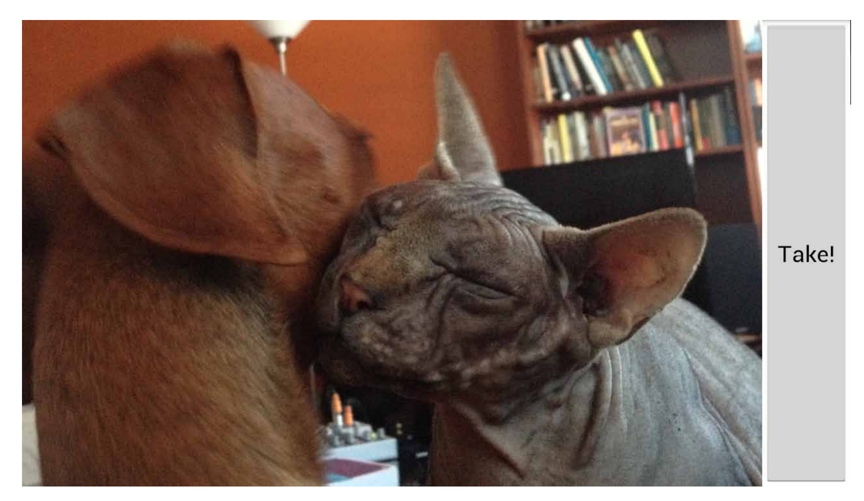


Рис. 19.1. Предварительное изображение с камеры

На рис. 19.2 изображены новые объекты, которые будут созданы в приложении.

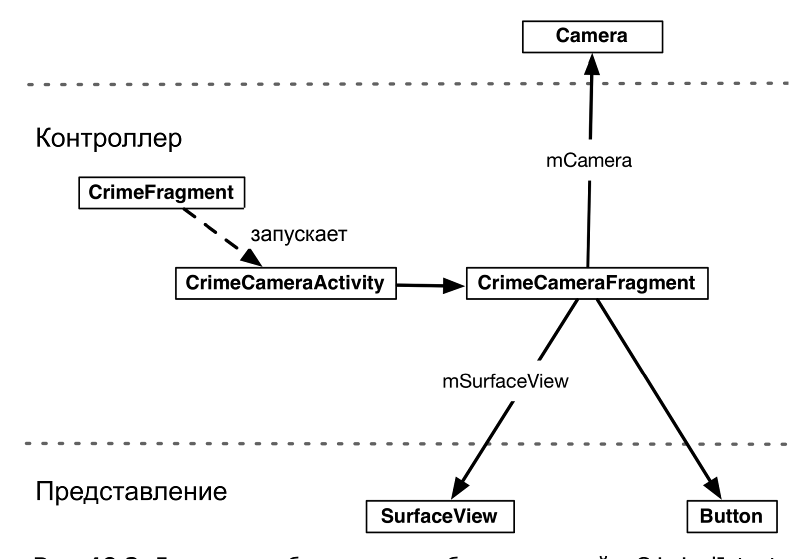


Рис. 19.2. Диаграмма объектов для работы с камерой в CriminalIntent

Экземпляр `Camera` предоставляет доступ к камере устройства на аппаратном уровне. Камера относится к ресурсам монопольного доступа; только одна активность может обращаться к камере.

Экземпляр SurfaceView будет выполнять функции видоискателя. SurfaceView — специальное представление для отображения контента прямо на экране.

Начнем с создания макета для представления CrimeCameraFragment, самого класса CrimeCameraFragment и класса CrimeCameraActivity. Затем мы организуем создание и управление видоискателем в CrimeCameraFragment. Наконец, в CrimeFragment будет включена поддержка запуска экземпляра CrimeCameraActivity.

Создание макета фрагмента

Создайте новый файл Android XML Layout с именем fragment_crime_camera.xml и назначьте его корневым элементом FrameLayout. Добавьте виджеты, изображенные на рис. 19.3.

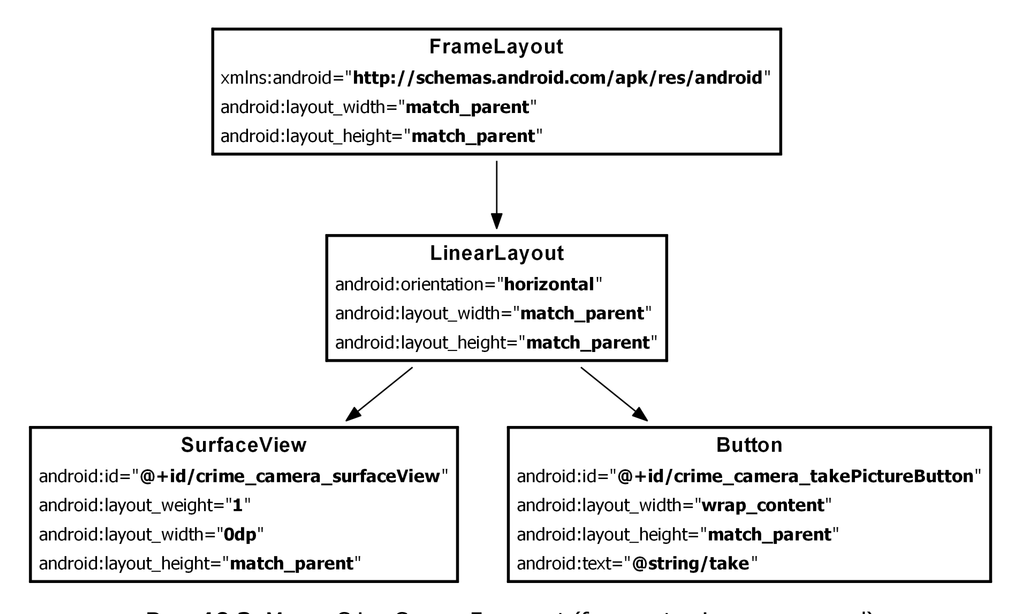


Рис. 19.3. Макет CrimeCameraFragment (fragment_crime_camera.xml)

В файле макета единственным потомком FrameLayout является виджет LinearLayout, из-за чего выводится предупреждение о бесполезности LinearLayout. Пока не обращайте внимания; мы назначим FrameLayout второе дочернее представление в главе 20.

В элементе LinearLayout для размещения дочерних представлений используется комбинация атрибутов layout_width и layout_weight. Место, необходимое для виджета Button, выделяется из-за наличия атрибута android:layout_width="wrap_content", а SurfaceView не достается ничего (android:layout_width="0dp"). При распределении оставшегося пространства только виджет SurfaceView имеет атрибут layout_weight, поэтому все оставшееся пространство достается SurfaceView.

На рис. 19.4 показано, как выглядит этот макет.

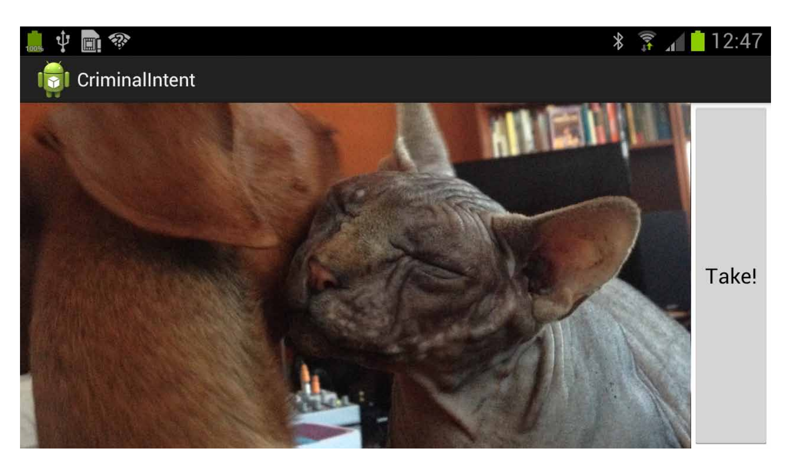


Рис. 19.4. Область предварительного просмотра и кнопка

В файле strings.xml добавьте строковый ресурс для текста кнопки.

Листинг 19.1. Добавление строки для кнопки камеры (strings.xml)

```
  ...
  <string name="show_subtitle">Show Subtitle</string>
  <string name="subtitle">Sometimes tolerance is not a virtue.</string>
  <string name="take">Take!</string>

</resources>
```

Создание класса CrimeCameraFragment

Создайте новый класс с именем `CrimeCameraFragment` и назначьте его суперклассом `android.support.v4.app.Fragment`. В файле CrimeCameraFragment.java добавьте новые поля, приведенные в листинге 19.2. Переопределите метод `onCreateView(…)` так, чтобы он заполнял макет и получал ссылки на виджеты. Пока назначьте кнопке слушателя, который просто завершает активность-хоста; при этом пользователь возвращается к предыдущему экрану.

Листинг 19.2. Исходный фрагмент камеры (CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    private Camera mCamera;
    private SurfaceView mSurfaceView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);
        Button takePictureButton = (Button)v
            .findViewById(R.id.crime_camera_takePictureButton);
```

```
        takePictureButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                getActivity().finish();
            }
        });

        mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);

        return v;
    }
}
```

Создание класса CrimeCameraActivity

Создайте новый субкласс SingleFragmentActivity с именем CrimeCameraActivity. Переопределите метод createFragment(), чтобы он возвращал новый экземпляр CrimeCameraFragment.

Листинг 19.3. Создание активности камеры (CrimeCameraActivity.java)

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```

Включение активности и разрешений камеры в манифест

На следующем шаге следует объявить CrimeCameraActivity в манифесте. Кроме того, следует запросить для приложения право на использование камеры, добавив в манифест элемент uses-permission.

Внесите в файл AndroidManifest.xml изменения, представленные в листинге 19.4.

Листинг 19.4. Добавление разрешений и активности камеры в манифест (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.criminalintent"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16"/>
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-feature android:name="android.hardware.camera" />

  <application

         ...
```

продолжение ⇗

Листинг 19.4 (*продолжение*)

```
    <activity android:name=".CrimeCameraActivity"
      android:screenOrientation="landscape"
      android:label="@string/app_name">
    </activity>

  </application>

</manifest>
```

Элемент uses-feature сообщает, что вашему приложению необходима указанная функция устройства. Значение android.hardware.camera гарантирует, что когда ваше приложение появится в Google Play, оно будет предлагаться только для устройств, оснащенных камерой.

Обратите внимание на то, что в объявлении активности мы принудительно заставляем активность использовать альбомный режим при помощи атрибута android:screenOrientation. Тем самым предотвращается изменение интерфейса активности в то время, как пользователь пытается подобрать правильный угол для снимка.

У атрибута screenOrientation неожиданно много возможных значений. Например, можно назначить активности такую же ориентацию, как у ее родителя, или выбрать альбомную ориентацию в любом направлении в зависимости от аппаратного датчика. За дополнительной информацией обращайтесь к документации элемента <activity>.

Использование API камеры

До настоящего момента мы занимались подготовительным созданием активностей. Пришло время заняться концепциями и классами, относящимися непосредственно к камере.

Открытие и освобождение камеры

Начнем с управления ресурсом камеры. Мы предоставили CrimeCameraFragment экземпляр Camera. Камера является важным общесистемным ресурсом, поэтому очень важно получать ее только тогда, когда она нужна, и освобождать сразу же после завершения работы. В противном случае камера окажется недоступной для других устройств до перезагрузки устройства.

Для управления камерой используются следующие методы класса Camera:

```
public static Camera open(int cameraId)
public static Camera open()
public final void release()
```

Метод open(int) появился в API уровня 9, а в API уровня 8 используется метод open() без параметров.

Методы обратного вызова точек жизненного цикла CrimeCameraFragment, в которых происходит открытие и освобождение камеры, называются onResume() и onPause().

Эти два метода обозначают границы, в пределах которых пользователь может взаимодействовать с представлением камеры. (Обратите внимание: `onResume()` вызывается тогда, когда фрагмент впервые появляется на экране.)

В методе `CrimeCameraFragment.onResume()` мы инициализируем камеру статическим методом `Camera.open(int)` и передаем 0, чтобы открыть первую камеру, доступную на устройстве. Обычно это камера на задней панели, но если она отсутствует на устройстве (как, например, на Nexus 7), открывается камера на передней панели.

В API уровня 8 вызывается метод `Camera.open()` без параметров. Чтобы защитить свой код для Froyo, проверьте версию устройства и вызовите `Camera.open()` в API уровня 8.

Листинг 19.5. Открытие камеры в методе onResume() (CrimeCameraFragment.java)

```
@TargetApi(9)
@Override
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}
```

(Android Lint может выдать предупреждения относительно открытия камеры в главном потоке. Эти предупреждения обоснованы, но мы не будем обращать на них внимания, пока не познакомимся поближе с многопоточностью в главе 26.)

Когда фрагмент уходит с экрана, следует освободить ресурс камеры, чтобы он стал доступным для других приложений. Соблюдайте правила хорошего тона и переопределите метод `onPause()` для освобождения камеры.

Листинг 19.6. Реализация методов жизненного цикла (CrimeCameraFragment.java)

```
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}

@Override
public void onPause() {
    super.onPause();
    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

Обратите внимание на проверку наличия экземпляра камеры перед вызовом `release()`. Всегда используйте ее перед выполнением кода камеры. Хотя вы выдали

запрос на ее использование, камера может оказаться недоступной — скажем, она может использоваться другой активностью, а на виртуальном устройстве камеры нет вообще. Если экземпляр камеры по какой-либо причине не существует, проверка null предотвратит сбой приложения.

SurfaceView, SurfaceHolder и Surface

SurfaceView предоставляет реализация интерфейса SurfaceHolder. В файле CrimeCameraFragment.java добавьте следующий код, чтобы получить объект SurfaceHolder для экземпляра SurfaceView.

Листинг 19.7. Получение SurfaceHolder (CrimeCameraFragment.java)

```
@Override
@SuppressWarnings("deprecation")
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {

    ...
    mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);
    SurfaceHolder holder = mSurfaceView.getHolder();
    // Метод setType() и константа SURFACE_TYPE_PUSH_BUFFERS считаются
    // устаревшими, но они необходимы для того, чтобы функция
    // предварительного просмотра изображения с камеры
    // Camera работала на устройствах до 3.0.
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    return v;
}
```

Метод setType(…) и константа SURFACE_TYPE_PUSH_BUFFERS считаются устаревшими, и компилятор выдает предупреждения об использовании устаревшего кода. Eclipse помечает устаревшие методы перечеркиванием.

Почему мы предлагаем использовать устаревший код? Метод setType(…) и константа SURFACE_TYPE_PUSH_BUFFERS необходимы для работы режима предварительного просмотра камеры на устройствах до Honeycomb.

В листинге 19.7 эти предупреждения подавляются аннотацией @SuppressWarnings. На первый взгляд решение кажется излишне бесцеремонным, но это лучшее решение проблемы устаревшего кода и совместимости в Android. Тема устаревшего кода в Android более подробно рассматривается в конце главы 20.

Объект SurfaceHolder обеспечивает связь с другим объектом — Surface. Объект *поверхности* Surface представляет буфер с низкоуровневыми данными пикселов.

Объект Surface имеет определенный жизненный цикл: он создается за вас при появлении SurfaceView на экране и уничтожается, когда объект SurfaceView перестает быть видимым. Следите за тем, чтобы в то время, когда Surface не существует, объект не использовался для графических операций.

В отличие от других объектов View, ни SurfaceView, ни один из его «партнеров» ничего не выводят на себе. *Клиентом* (client) Surface называется объект, который

хочет что-то вывести в его буфер. В классе `CrimeCameraFragment` клиентом является экземпляр `Camera`.

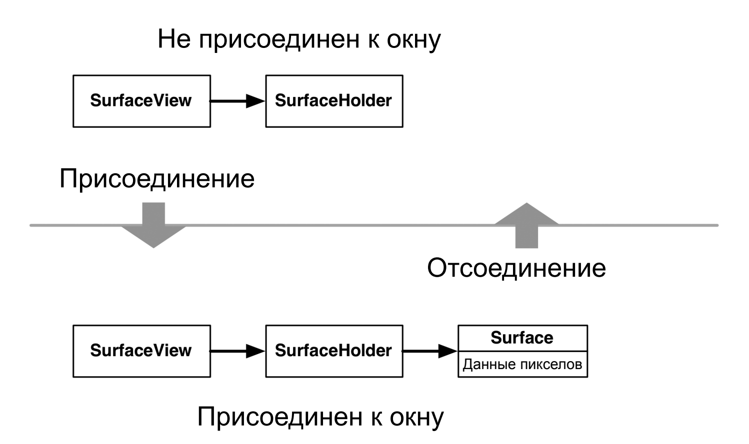


Рис. 19.5. SurfaceView, SurfaceHolder и Surface

Не забывайте: вы должны следить за тем, чтобы на объекте `Surface` ничего не рисовалось, когда его нет. Чтобы достичь такого положения дел (рис. 19.6), необходимо присоединять экземпляр `Camera` к `SurfaceHolder` при создании `Surface` и отсоединять его при уничтожении `Surface`.



Рис. 19.6. Как должно быть

Класс `SurfaceHolder` предоставляет другой интерфейс, который позволяет это сделать — `SurfaceHolder.Callback`. Этот интерфейс прослушивает события жизненного цикла `Surface` для координации `Surface` с клиентом.

Интерфейс `SurfaceHolder.Callback` содержит три метода:

```
public abstract void surfaceCreated(SurfaceHolder holder)
```

Метод вызывается при выводе на экран иерархии, к которой принадлежит `SurfaceView`. Здесь `Surface` связывается со своим клиентом.

```
public abstract void surfaceChanged(SurfaceHolder holder, int format, int width,
int height)
```

Когда поверхность выводится впервые, вызывается метод `surfaceChanged(…)`. Вызов передает информацию о формате пикселов, а также ширине и высоте поверхности. В этом методе вы сообщаете клиенту `Surface` размер области вывода.

```
public abstract void surfaceDestroyed(SurfaceHolder holder)
```

При удалении `SurfaceView` с экрана объект `Surface` уничтожается. В этом методе вы приказываете клиенту `Surface` прекратить использование `Surface`.

Далее перечислены методы `Camera`, которые будут использоваться для обработки событий жизненного цикла `Surface`:

```
public final void setPreviewDisplay(SurfaceHolder holder)
```

Метод связывает камеру с `Surface`. Он будет вызываться в `surfaceCreated()`.

```
public final void startPreview()
```

Метод начинает прорисовку кадров на `Surface`. Он будет вызываться в `surfaceChanged()`.

```
public final void stopPreview()
```

Метод прекращает прорисовку кадров на `Surface`. Он будет вызываться в `surfaceDestroyed()`.

Добавьте в файл CrimeCameraFragment.java реализацию `SurfaceHolder.Callback` для координации жизненного цикла `Surface` с предварительным просмотром изображения с камеры.

Листинг 19.8. Реализация SurfaceHolder.Callback (CrimeCameraFragment.java)

```
...
SurfaceHolder holder = mSurfaceView.getHolder();
// Метод setType() и константа SURFACE_TYPE_PUSH_BUFFERS считаются
// устаревшими, но они необходимы для того, чтобы функция
// предварительного просмотра изображения с камеры
// Camera работала на устройствах до 3.0.
holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

holder.addCallback(new SurfaceHolder.Callback() {

    public void surfaceCreated(SurfaceHolder holder) {
        // Приказываем камере использовать указанную
        // поверхность как область предварительного просмотра
        try {
```

```
                if (mCamera != null) {
                    mCamera.setPreviewDisplay(holder);
                }
            } catch (IOException exception) {
                Log.e(TAG, "Error setting up preview display", exception);
            }
        }

        public void surfaceDestroyed(SurfaceHolder holder) {
            // Дальнейший вывод на поверхности невозможен,
            // прекращаем предварительный просмотр.
            if (mCamera != null) {
                mCamera.stopPreview();
            }
        }

        public void surfaceChanged(SurfaceHolder holder, int format, int w, int h)
{
            if (mCamera == null) return;

            // Размер поверхности изменился; обновить размер
            // области предварительного просмотра камеры
            Camera.Parameters parameters = mCamera.getParameters();
            Size s = null; // Будет изменено в следующем разделе
            parameters.setPreviewSize(s.width, s.height);
            mCamera.setParameters(parameters);
            try {
                mCamera.startPreview();
            } catch (Exception e) {
                Log.e(TAG, "Could not start preview", e);
                mCamera.release();
                mCamera = null;
            }
        }
    });

    return v;
}
```

Если запустить предварительный просмотр не удалось, камера освобождается. Каждое открытие камеры должно сопровождаться ее освобождением — даже при возникновении исключения.

В реализации `surfaceChanged(…)` размер области предварительного просмотра камеры задается равным `null`. Это временное значение действует до того момента, когда мы сможем определить допустимый размер области. Задать произвольный размер нельзя, при недопустимом значении выдается исключение.

Определение размера области предварительного просмотра

Первым шагом станет получение списка допустимых размеров области предварительного просмотра камеры от вложенного класса `Camera.Parameters`, который включает следующий метод:

```
public List<Camera.Size> getSupportedPreviewSizes()
```

Метод возвращает список экземпляров класса android.hardware.Camera.Size, в котором упакованы значения ширины и высоты изображения.

Сравнивая размеры в списке с шириной и высотой объекта Surface, переданного surfaceChanged(…), можно подобрать размер, наиболее подходящий для вашего объекта Surface.

Включите в CrimeCameraFragment следующий метод, который получает список размеров и выбирает размер с наибольшим количеством пикселов. Код не слишком элегантен, но он хорошо подойдет для наших целей.

Листинг 19.9. Выбор оптимального поддерживаемого размера (CrimeCameraFragment.java)

```
/** Простой алгоритм для получения наибольшего доступного размера.
 * Более мощная версия представлена в файле CameraPreview.java
 * приложения-примера ApiDemos от Android. */
private Size getBestSupportedSize(List<Size> sizes, int width, int height) {
    Size bestSize = sizes.get(0);
    int largestArea = bestSize.width * bestSize.height;
    for (Size s : sizes) {
        int area = s.width * s.height;
        if (area > largestArea) {
            bestSize = s;
            largestArea = area;
        }
    }
    return bestSize;
}
```

Этот метод вызывается для задания размера области предварительного просмотра в surfaceChanged(…).

Листинг 19.10. Вызов getBestSupportedSize(…) (CrimeCameraFragment.java)

```
...
holder.addCallback(new SurfaceHolder.Callback() {

    ...

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // Размер поверхности изменился; обновить размер
        // области предварительного просмотра камеры
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = null;
        Size s = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
        parameters.setPreviewSize(s.width, s.height);
        mCamera.setParameters(parameters);
        try {
            mCamera.startPreview();
        } catch (Exception e) {
            Log.e(TAG, "Could not start preview", e);
            mCamera.release();
            mCamera = null;
        }
    }
});
```

Запуск CrimeCameraActivity из CrimeFragment

Чтобы видоискатель заработал, мы добавим в `CrimeFragment` кнопку. При нажатии этой кнопки `CrimeFragment` запускает экземпляр `CrimeCameraActivity`. Как должно выглядеть представление фрагмента `CrimeFragment`, показано на рис. 19.7.

Чтобы виджеты были расположены так, как показано на рис. 19.7, необходимо добавить три виджета `LinearLayout` и `ImageButton`.

На рис. 19.8 представлены изменения, которые необходимо внести в макет `CrimeFragment` по умолчанию.

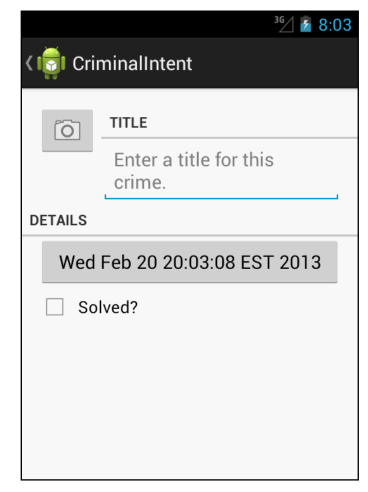


Рис. 19.7. CrimeFragment с кнопкой камеры и сдвинутыми виджетами

LinearLayout

LinearLayout
android:layout_width="**fill_parent**"
android:layout_height="**wrap_content**"
android:orientation="**horizontal**"
android:layout_marginLeft="**16dp**"
android:layout_marginTop="**16dp**"

TextView

Button

CheckBox

LinearLayout
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:orientation="**vertical**"
android:layout_marginRight="**4dp**"

LinearLayout
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:orientation="**vertical**"
android:layout_weight="**1**"

ImageButton
android:id="**@+id/crime_imageButton**"
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"
android:src="**@android:drawable/ic_menu_camera**"

TextView

EditText

Рис. 19.8. Добавление кнопки камеры и реорганизация представления (layout/fragment_crime.xml)

Внесите аналогичные изменения в альбомном макете.

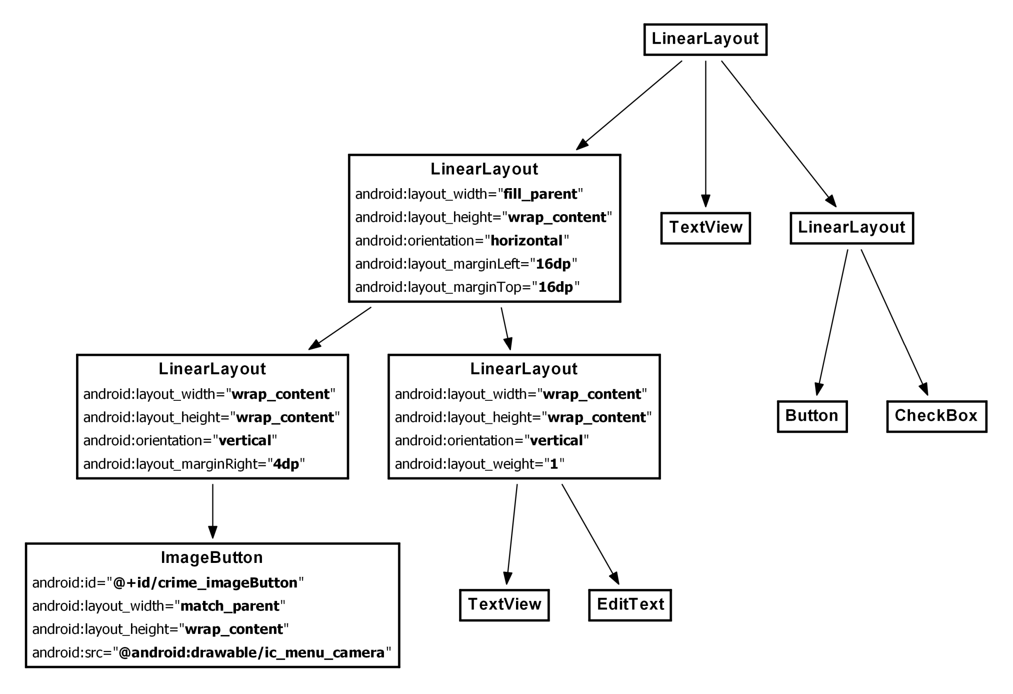


Рис. 19.9. Добавление кнопки камеры и реорганизация представления (layout-land/fragment_crime.xml)

Добавьте в класс `CrimeFragment` поле для кнопки `ImageButton`, получите ссылку на кнопку и назначьте слушателя `OnClickListener`, запускающего `CrimeCameraActivity`.

Листинг 19.11. Запуск CrimeCameraActivity (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private ImageButton mPhotoButton;

    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        ...

        mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivity(i);
            }
        });

        return v;
    }
```

Если на устройстве нет камеры, кнопка `mPhotoButton` должна быть заблокирована. Присутствие камеры проверяется запросом к `PackageManager`. Добавьте следующий код в метод `onCreateView(…)`, чтобы заблокировать `ImageButton` при отсутствии камеры на устройстве.

Листинг 19.12. Проверка наличия камеры (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    ...

    mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        ...
    });

    // Если камера недоступна, заблокировать функциональность
    // работы с камерой
    PackageManager pm = getActivity().getPackageManager();
    if (!pm.hasSystemFeature(PackageManager.FEATURE_CAMERA) &&
            !pm.hasSystemFeature(PackageManager.FEATURE_CAMERA_FRONT)) {
        mPhotoButton.setEnabled(false);
    }

    ...
}
```

После получения `PackageManager` мы вызываем `hasSystemFeature(String)`, передавая константы интересующих нас функций устройства. Обратите внимание: константа `FEATURE_CAMERA` обозначает камеру на задней панели, а `FEATURE_CAMERA_FRONT` — камеру на передней панели. Если устройство не оснащено ни одной из этих камер, кнопка `ImageButton` блокируется.

Запустите приложение CriminalIntent. Откройте преступление и нажмите кнопку камеры. На экране появляется «живое» изображение предварительного просмотра. Нажатие кнопки Take! возвращает к представлению `CrimeFragment`.

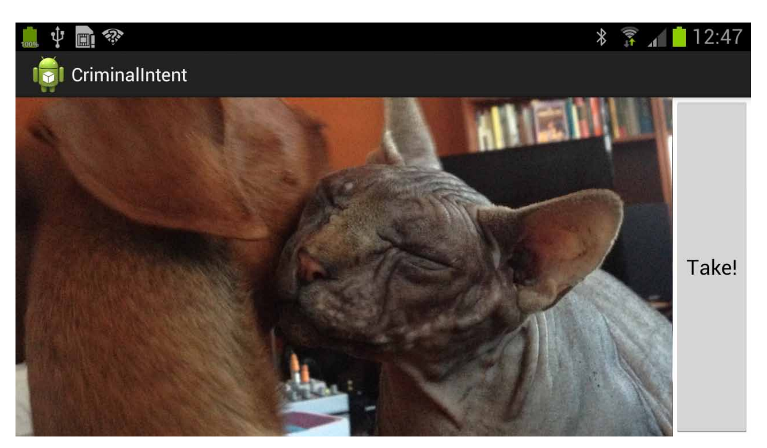


Рис. 19.10. Предварительный просмотр изображения с камеры

Мы видим последствия принудительного назначения активности альбомной ориентации в манифесте. Даже если устройство находится в книжной ориентации, текст кнопки фиксируется в альбомном положении.

Скрытие панели состояния и панели действий

На рис. 19.10 панель действий активности и панель состояния скрывают часть области предварительного просмотра. В этой активности внимание пользователя в основном сосредоточено на видоискателе, поэтому эти элементы только занимают место на экране. Лучше убрать их из представления активности.

Интересно, что скрыть панель состояния и панель действий из CrimeCameraFragment нельзя; это необходимо делать в CrimeCameraActivity. Откройте файл CrimeCameraActivity.java и добавьте следующий код в метод onCreate(Bundle).

Листинг 19.13. Настройка активности (CrimeCameraActivity.java)

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Скрытие заголовка окна.
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Скрытие панели состояния и прочего оформления уровня ОС
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
        super.onCreate(savedInstanceState);
    }

    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```

Почему это необходимо делать в активности? Вызовы requestWindowFeature(…) и addFlags(…) должны быть сделаны до создания представления активности

в `Activity.setContentView(…)`, которое в `CrimeCameraActivity` вызывается в реализации `onCreate(Bundle)` суперкласса. Фрагмент не может быть добавлен до создания представления его активности-хоста. Следовательно, эти методы должны вызываться из активности.

Запустите приложение CriminalIntent и оцените увеличенную область предварительного просмотра.

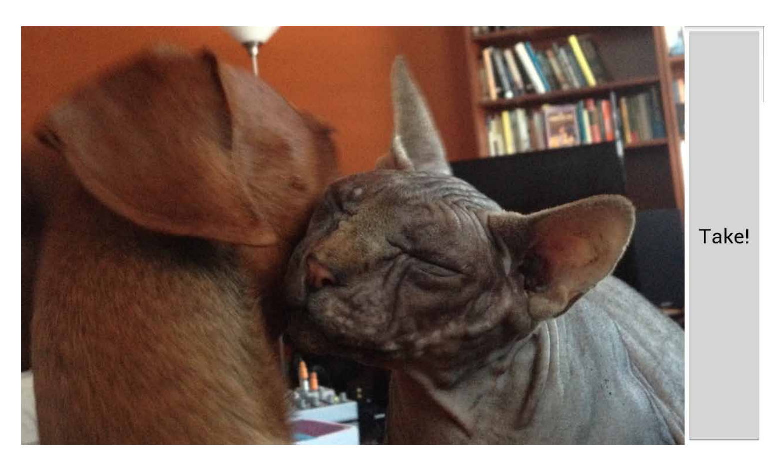


Рис. 19.11. Активность без панели состояния и панели действий

В следующей главе мы подробнее рассмотрим API работы с камерой, сохраним изображение в локальной файловой системе и выведем его в представлении `CrimeFragment`.

Для любознательных: запуск активностей из командной строки

В этой главе нам удалось довольно быстро протестировать код работы с камерой, добавив кнопку в `CrimeFragment` и запустив `CrimeCameraActivity`. Однако иногда требуется протестировать код новой активности до того, как он будет связан с остальным кодом приложения.

Простой способ тестирования «на скорую руку» основан на замене фильтра интента для лаунчера в манифесте — объявление основной активности лаунчера заменяется объявлением нужной вам активности. При запуске приложения на экране появляется ваша активность. С другой стороны, вы не сможете получить доступ к другим частям приложения, пока в манифесте не будет восстановлен порядок.

Подмена интента для лаунчера иногда оказывается неуместной — например, если вы работаете над приложением еще с пятью людьми. В таких ситуациях существует более правильное решение: запуск активности из командной строки программой *adb*.

Прежде всего активность следует экспортировать. Добавьте следующий атрибут в объявление `CrimeCameraActivity` в файле AndroidManifest.xml:

```
<activity android:name=".CrimeCameraActivity"
  android:exported="true"
  android:screenOrientation="landscape"
  android:label="@string/app_name">
</activity>
```

По умолчанию активности могут запускаться только из вашего приложения. Присваивая атрибуту `exported` значение `true`, вы сообщаете Android, что данная активность также может запускаться другими приложениями. (При добавлении фильтра интентов в объявление активности атрибуту `exported` автоматически задается значение `true`.)

Затем найдите утилиту *adb* в папке platform-tools своей установки Android SDK. Мы рекомендуем добавить папки platform-tools и tools в список путей командного процессора.

Любители командной строки будут в восторге от *adb* (Android Debug Bridge). Программа *adb* может использоваться для выполнения многих операций, для которых обычно используется Eclipse. Вы можете отслеживать данные LogCat, открыть окно командного процессора для устройства, просматривать файловую систему, загружать и отправлять файлы, получать списки подключенных устройств, выполнять сброс — это отличный инструмент.

Программа *adb* может использоваться с несколькими устройствами, но с ней проще работать при наличии только одного устройства. Закройте или отключите все дополнительные эмуляторы или устройства и запустите CriminalIntent на тестовом устройстве, как обычно. Затем используйте следующий вызов для запуска `CrimeCameraActivity` из командной строки:

```
$ adb shell am start -n com.bignerdranch.android.criminalintent/.
CrimeCameraActivity
Starting: Intent { cmp=com.bignerdranch.android.criminalintent/.CrimeCameraActivity
}
```

Выполните команду, и `CrimeCameraActivity` заработает на вашем эмуляторе или устройстве.

Эта команда представляет собой сокращенную запись для выполнения команды в командном процессоре на устройстве. Она эквивалентна следующей последовательности:

```
$ adb shell
shell@android:/ $ am start -n com.bignerdranch.android.criminalintent/.
CrimeCameraAct\
ivity
```

Команда *am* — находящаяся на вашем устройстве программа, которая называется *менеджером активностей* (Activity Manager). Она позволяет запускать и останавливать компоненты Android, а также отправлять интенты из командной строки. Полная информация о возможностях *am* выводится командой **adb shell am**.

20 Камера II: Съемка и обработка изображений

В этой главе мы получим изображение с камеры и сохраним его в формате JPEG в файловой системе, после чего свяжем файл JPEG с `Crime` и отобразим его в представлении `CrimeFragment`.

Также пользователю будет предложена возможность просмотра увеличенной версии изображения в `DialogFragment`.

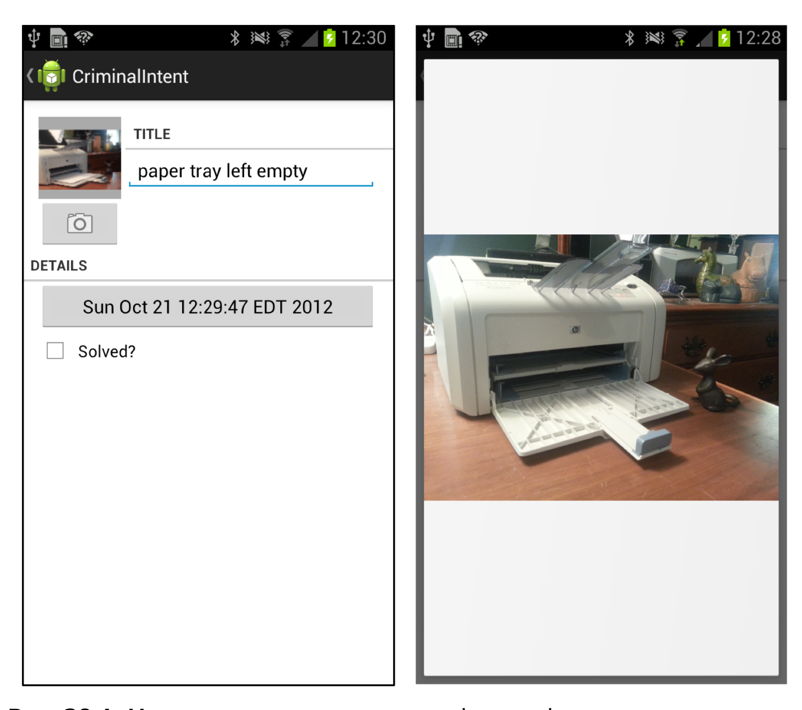


Рис. 20.1. Миниатюра и полноразмерная фотография места преступления

Получение снимка

Начнем с обновления макета `CrimeCameraFragment` — в него будет добавлен индикатор прогресса. Съемка может занять некоторое время, и мы не хотим, чтобы пользователи начинали нервничать.

Добавьте в файл layout/fragment_crime_camera.xml виджеты `FrameLayout` и `ProgressBar`, показанные на рис. 20.2.

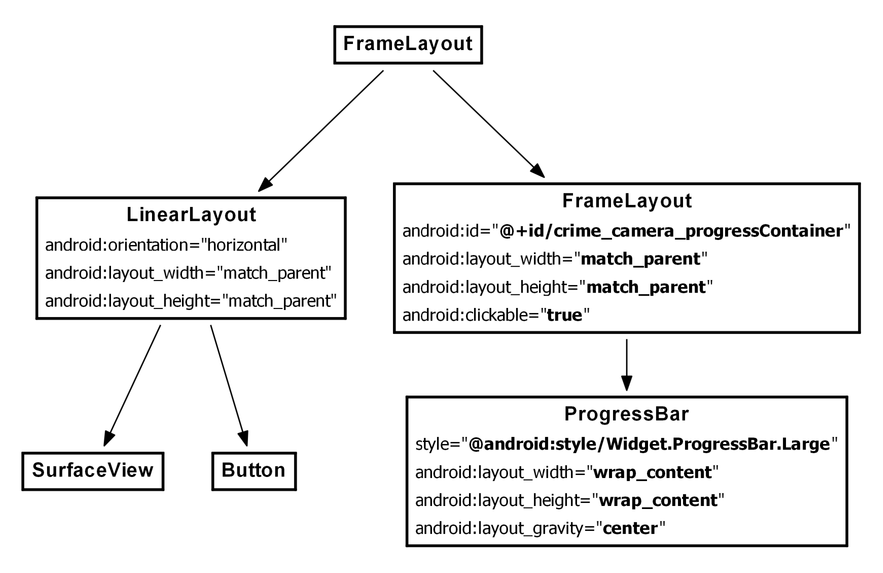


Рис. 20.2. Добавление виджетов FrameLayout и ProgressBar (fragment_crime_camera.xml)

Стиль `@android:style/Widget.ProgressBar.Large` создает большой круговой индикатор неопределенной продолжительности.

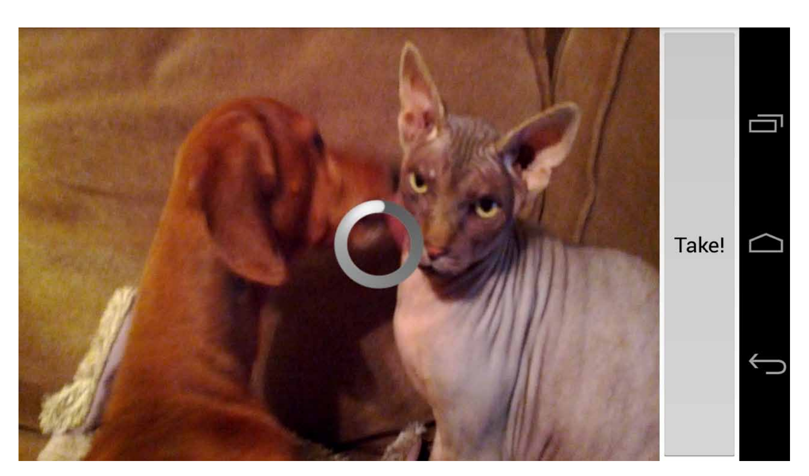


Рис. 20.3. Круговой индикатор прогресса

Виджет FrameLayout (и его потомок ProgressBar) изначально невидимы. Они становятся видимыми только после того, как пользователь нажмет кнопку Take! и начнется процесс съемки.

Обратите внимание: ширина и высота FrameLayout определяются значением match_parent. Корневой элемент FrameLayout накладывает свои дочерние представления поверх друг друга в порядке их определения. Таким образом, виджет FrameLayout, содержащий ProgressBar, полностью закроет своего соседа LinearLayout.

Когда виджет FrameLayout становится видимым, пользователь по-прежнему сможет видеть почти все содержимое LinearLayout. Только виджет ProgressBar будет скрывать часть изображения. Однако определение ширины и высоты FrameLayout в режиме match_parent и задание свойства android:clickable="true" гарантирует, что FrameLayout будет перехватывать все события касаний (и ничего не делать с ними). Тем самым предотвращается взаимодействие пользователя с содержимым LinearLayout, и в частности повторное нажатие кнопки Take!.

Вернитесь к файлу CrimeCameraFragment.java. Добавьте поле для виджета FrameLayout, получите ссылку на виджет и пометьте его как невидимый.

Листинг 20.1. Подключение FrameLayout (CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {
    ...
    private View mProgressContainer;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);

        mProgressContainer = v.findViewById(R.id.crime_camera_progressContainer);
        mProgressContainer.setVisibility(View.INVISIBLE);

        ...

        return v;
    }
...
}
```

Реализация обратных вызовов камеры

Итак, индикатор прогресса работает; можно переходить к захвату кадра с видоискателя и сохранению его в формате JPEG. Для этого используется следующий метод класса Camera:

```
public final void takePicture(Camera.ShutterCallback shutter,
    Camera.PictureCallback raw,
    Camera.PictureCallback jpeg)
```

Вызов ShutterCallback происходит при захвате изображения с камеры, но до того, как данные изображения будут обработаны и станут доступными. Первый вызов

`PictureCallback` происходит при появлении необработанных графических данных и обычно используется для их предварительной обработки перед сохранением. Второй вызов `PictureCallback` происходит при появлении доступной JPEG-версии изображения..

Вы можете написать реализации этих интерфейсов и передать их `takePicture(…)`. Если какой-либо метод обратного вызова не используется, передайте `null` в соответствующем параметре `takePicture(…)`.

В CriminalIntent мы реализуем `ShutterCallback` и метод обратного вызова для изображения JPEG.

На рис. 20.4 изображена схема взаимодействий между объектами.

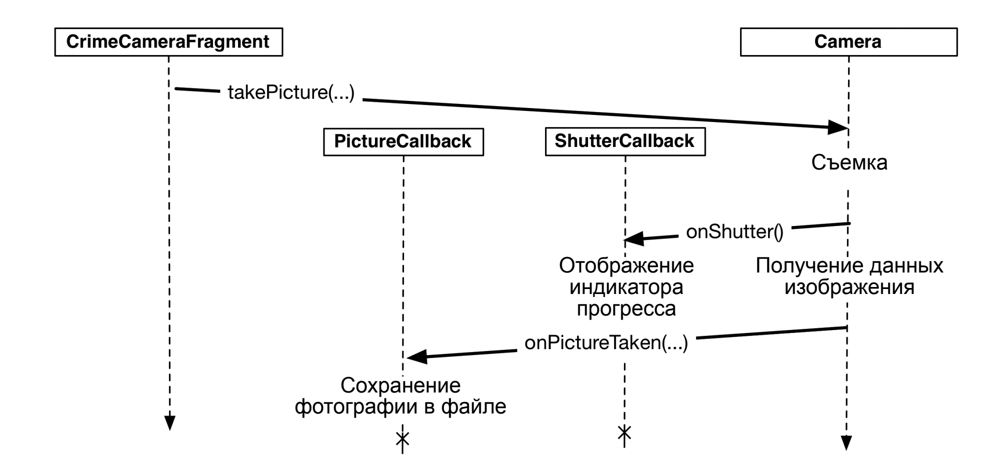


Рис. 20.4. Получение фотографии в CrimeCameraFragment

В схеме задействованы два интерфейса, каждый из которых содержит один метод.

```
public static interface Camera.ShutterCallback {
    public abstract void onShutter();
}
public static interface Camera.PictureCallback {
    public abstract void onPictureTaken (byte[] data, Camera camera);
}
```

Добавьте в файл CrimeCameraFragment.java реализацию `Camera.ShutterCallback`, которая отображает индикатор прогресса, и реализацию `Camera.PictureCallback`, которая обрабатывает выбор имени и сохранение файла JPEG.

Листинг 20.2. Реализация обратных вызовов в takePicture(…) (CrimeCameraFragment.java)

```
...
private View mProgressContainer;

private Camera.ShutterCallback mShutterCallback = new Camera.ShutterCallback() {
    public void onShutter() {
```

```
        // Отображение индикатора прогресса
        mProgressContainer.setVisibility(View.VISIBLE);
    }
};

private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
        // Создание имени файла
        String filename = UUID.randomUUID().toString() + ".jpg";
        // Сохранение данных jpeg на диске
        FileOutputStream os = null;
        boolean success = true;
        try {
            os = getActivity().openFileOutput(filename, Context.MODE_PRIVATE);
            os.write(data);
        } catch (Exception e) {
            Log.e(TAG, "Error writing to file " + filename, e);
            success = false;
        } finally {
            try {
                if (os != null)
                    os.close();
            } catch (Exception e) {
                Log.e(TAG, "Error closing file " + filename, e);
                success = false;
            }
        }

        if (success) {
            Log.i(TAG, "JPEG saved at " + filename);
        }
        getActivity().finish();
    }
};
...
```

В методе onPictureTaken(…) создается уникальная строка, которая используется для имени файла. Затем классы ввода-вывода Java используются для открытия выходного потока и записи данных JPEG, полученных от Camera. Если операция прошла нормально, в журнале регистрируется уведомление об успехе.

Обратите внимание: мы не возвращаем mProgressContainer в невидимое состояние. Это не нужно, потому что в обратном вызове PictureCallback активность завершается, а представление уничтожается.

После подготовки обратных вызовов измените слушателя кнопки Take!, чтобы он вызывал takePicture(…). Передайте null вместо обратного вызова необработанных данных, который мы не реализовали.

Листинг 20.3. Вызов takePicture(…) по щелчку на кнопке (CrimeCameraFragment.java)

```
@Override
@SuppressWarnings("deprecation")
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...
```

продолжение ⇗

Листинг 20.3 (*продолжение*)

```
    takePictureButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            getActivity().finish();
            if (mCamera != null) {
                mCamera.takePicture(mShutterCallback, null, mJpegCallback);
            }
        }
    });

    ...

    return v;
}
```

Назначение размера изображения

Камера должна знать размер создаваемого изображения. Размер фотографии задается так же, как размер области предварительного просмотра. Для получения списка допустимых размеров изображения используется метод класса `Camera.Parameters`:

```
public List<Camera.Size> getSupportedPictureSizes()
```

В методе `surfaceChanged(…)` мы используем метод `getBestSupportedSize(…)` для определения поддерживаемого размера изображения, соответствующего вашему экземпляру `Surface`. Далее задается размер изображения камеры.

Листинг 20.4. Вызов getBestSupportedSize(…) для задания размера изображения (CrimeCameraFragment.java)

```
    ...

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        if (mCamera == null) return;

        // Размер поверхности изменился; обновить размер области
        // предварительного просмотра камеры
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
        parameters.setPreviewSize(s.width, s.height);
        s = getBestSupportedSize(parameters.getSupportedPictureSizes(), w, h);
        parameters.setPictureSize(s.width, s.height);
        mCamera.setParameters(parameters);

        ...
    }
});
```

Запустите приложение CriminalIntent и нажмите кнопку Take!. Чтобы увидеть, где оказалось изображение, создайте в LogCat фильтр с тегом `CrimeCameraFragment`.

В этой точке `CrimeCameraFragment` предоставляет пользователю возможность сохранить изображение, полученное от камеры. Наша работа с API камеры закончена. В оставшейся части главы мы займемся классом `CrimeFragment` и интеграцией фотографий с остальными частями приложения.

Передача данных CrimeFragment

Чтобы предоставить `CrimeFragment` доступ к фотографиям, `CrimeCameraFragment` будет возвращать имя файла. На рис. 20.5 изображена последовательность событий взаимодействия между `CrimeFragment` и `CrimeCameraFragment`.

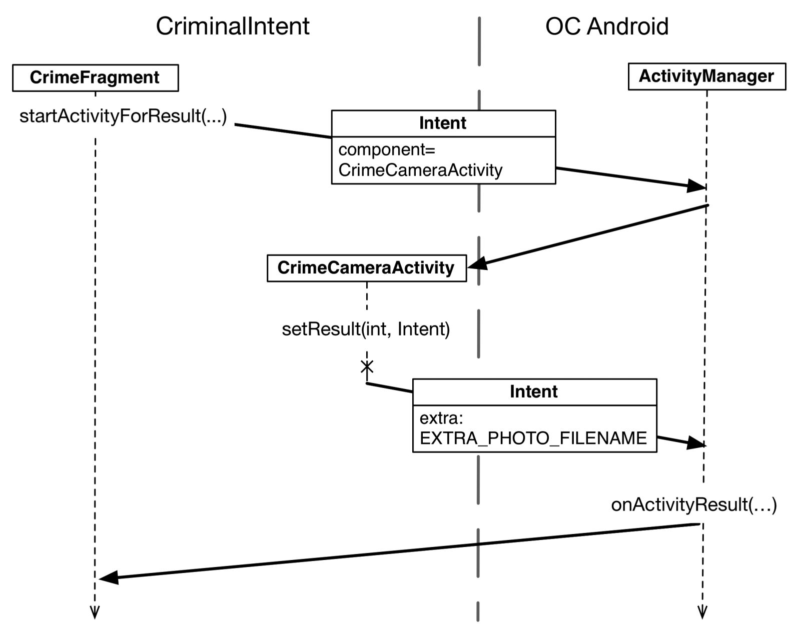


Рис. 20.5. CrimeCameraActivity назначает результат и дополнение

Сначала `CrimeFragment` запускает `CrimeCameraActivity` с возвращением результата. При получении изображения `CrimeCameraFragment` создает интент с дополнением, содержащим имя файла, и вызывает `setResult(…)`. `ActivityManager` передает интент `CrimePagerActivity` в методе `onActivityResult(…)`. Далее экземпляр `FragmentManager`, принадлежащий `CrimePagerActivity`, передает интент `CrimeFragment` в `CrimeFragment.onActivityResult(…)`.

Запуск CrimeCameraActivity с возвращением результата

В текущей версии `CrimeFragment` просто запускает `CrimeCameraActivity`.В файле CrimeFragment.java добавьте константу для кода запроса, а затем измените слушателя кнопки камеры, чтобы он запускал `CrimeCameraActivity` с возвращением результата.

Листинг 20.5. Запуск CrimeCameraActivity с возвращением результата (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private static final int REQUEST_DATE = 0;
    private static final int REQUEST_PHOTO = 1;
    ...
```

продолжение ⇗

Листинг 20.5 (*продолжение*)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...

    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            // Запуск активности камеры
            Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
            startActivity(i);
            startActivityForResult(i, REQUEST_PHOTO);
        }
    });
    ...
}
```

Назначение результата в CrimeCameraFragment

CrimeCameraFragment помещает имя файла в дополнение интента и передает его при вызове CrimeCameraActivity.setResult(int, Intent). В файле CrimeCameraFragment.java добавьте константу для дополнения, затем в обратном вызове изображения создайте интент и задайте код результата RESULT_OK, если изображение было сохранено, или RESULT_CANCELED, если возникли какие-то проблемы.

Листинг 20.6. Создание дополнения для имени файла с фотографией (CrimeCameraFragment.java)

```
public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    public static final String EXTRA_PHOTO_FILENAME =
        "com.bignerdranch.android.criminalintent.photo_filename";

    ...

    private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
        public void onPictureTaken(byte[] data, Camera camera) {
            ...
            try {
                ...
            } catch (Exception e) {
                ...
            } finally {
                ...
            }
            Log.i(TAG, "JPEG saved at " + filename);
            // Имя файла фотографии записывается в интент результата
            if (success) {
                Intent i = new Intent();
                i.putExtra(EXTRA_PHOTO_FILENAME, filename);
                getActivity().setResult(Activity.RESULT_OK, i);
            } else {
```

```
                getActivity().setResult(Activity.RESULT_CANCELED);
            }
            getActivity().finish();
        }
    };
```

Получение имени файла в CrimeFragment

В будущем CrimeFragment будет использовать имя файла для обновления уровней модели и представления CriminalIntent.

А пока в файле CrimeFragment.java просто переопределите метод onActivityResult(…), чтобы он проверял результат, получал имя файла и сохранял в журнале сообщение. Нам также понадобится метка TAG для CrimeFragment, облегчающая поиск в журнале.

Листинг 20.7. Получение имени файла (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private static final String TAG = "CrimeFragment"
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";
    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;

        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        } else if (requestCode == REQUEST_PHOTO) {
            // Создание нового объекта Photo и связывание его с Crime
            String filename = data
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename != null) {
                Log.i(TAG, "filename: " + filename);
            }
        }
    }
```

Запустите приложение CriminalIntent и сделайте снимок в CrimeCameraActivity. Проверьте данные LogCat и убедитесь в том, что экземпляр CrimeFragment получил имя файла.

Теперь, когда CrimeFragment знает имя файла, необходимо проделать немалую работу:

- Обновление уровня модели: мы напишем класс Photo, инкапсулирующий имя файла изображения. Также в класс Crime будет добавлено свойство mPhoto типа Photo. CrimeFragment использует имя файла для создания объекта Photo и задания свойства mPhoto класса Crime.

- Обновление представления CrimeFragment: мы добавим в макет CrimeFragment виджет ImageView и выведем на нем миниатюру фотографии Crime.
- Для вывода увеличенной версии изображения мы создадим субкласс DialogFragment с именем ImageFragment и передадим ему путь к отображаемой фотографии.

Обновление уровня модели

На рис. 20.6 изображена схема отношений между CrimeFragment, Crime и Photo.

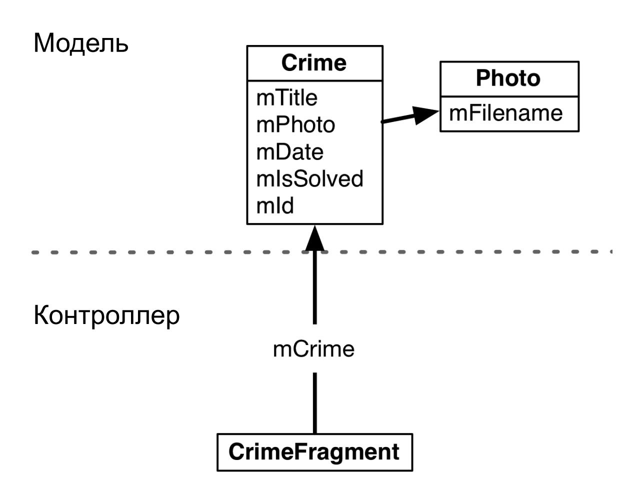


Рис. 20.6. Объекты модели и CrimeFragment

Добавление класса Photo

Создайте новый класс в пакете com.bignerdranch.android.criminalintent. Присвойте ему имя Photo и оставьте ему суперкласс java.lang.Object.

Добавьте в файл Photo.java поля и методы, приведенные в листинге 20.8.

Листинг 20.8. Класс Photo (Photo.java)

```
...
public class Photo {
    private static final String JSON_FILENAME = "filename";

    private String mFilename;
    /** Создание объекта Photo, представляющего файл на диске */
    public Photo(String filename) {
        mFilename = filename;
    }
```

```
    public Photo(JSONObject json) throws JSONException {
        mFilename = json.getString(JSON_FILENAME);
    }
    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_FILENAME, mFilename);
        return json;
    }
    public String getFilename() {
        return mFilename;
    }
}
```

Обратите внимание: у класса `Photo` два конструктора. Первый конструктор создает объект `Photo` по заданному имени файла, а второй представляет собой метод сериализации JSON, который используется классом `Crime` для сохранения и загрузки свойства типа `Photo`.

Включение в Crime свойства для хранения фотографии

Включите в класс `Crime` поле для объекта `Photo`, которое будет сериализоваться в формат JSON (листинг 20.9).

Листинг 20.9. Определение поля Photo в объекте Crime (Crime.java)

```
public class Crime {
    ...
    private static final String JSON_DATE = "date";
    private static final String JSON_PHOTO = "photo";

    ...
    private Date mDate = new Date();
    private Photo mPhoto;

    ...

    public Crime(JSONObject json) throws JSONException {
        ...
        mDate = new Date(json.getLong(JSON_DATE));
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        json.put(JSON_DATE, mDate.getTime());
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
        return json;
    }
```

Листинг 20.9 (*продолжение*)

```
    ...

    public Photo getPhoto() {
        return mPhoto;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }
}
```

Сохранение ссылки на фотографию

В файле CrimeFragment.java внесите изменения в метод onActivityResult(…), чтобы он создавал новый экземпляр Photo и связывал его с текущим экземпляром Crime.

Листинг 20.10. Создание экземпляра Photo

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;

    if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        } else if (requestCode == REQUEST_PHOTO) {
        // Создание нового объекта Photo и связывание его с Crime
        String filename = data
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename != null) {
            Log.i(TAG, "filename: " + filename);

            Photo p = new Photo(filename);
            mCrime.setPhoto(p);
            Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");
        }
    }
}
```

Запустите приложение CriminalIntent и сделайте снимок. По данным LogCat убедитесь в том, что к Crime теперь прилагается фотография.

Почему мы создаем класс Photo вместо того, чтобы просто добавить в Crime свойство для имени файла? Такое решение сработает в нашей ситуации, но нельзя исключать, что нам потребуется от объекта Photo что-то еще — например, вывод заголовка или обработка события касания. Для таких случаев нужен отдельный класс.

Обновление представления CrimeFragment

От обновления уровня модели можно перейти к обновлению уровня представления. Класс CrimeFragment будет отображать миниатюру фотографии в виджете ImageView.

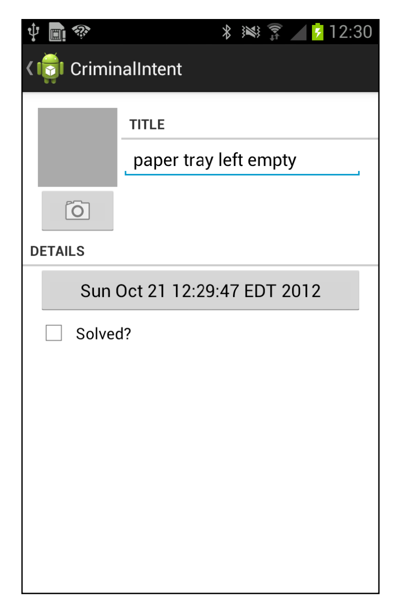


Рис. 20.7. CrimeFragment с новым виджетом ImageView

Добавление ImageView

Вернитесь к файлу layout/fragment_crime.xml и добавьте виджет `ImageView`, показанный на рис. 20.8.

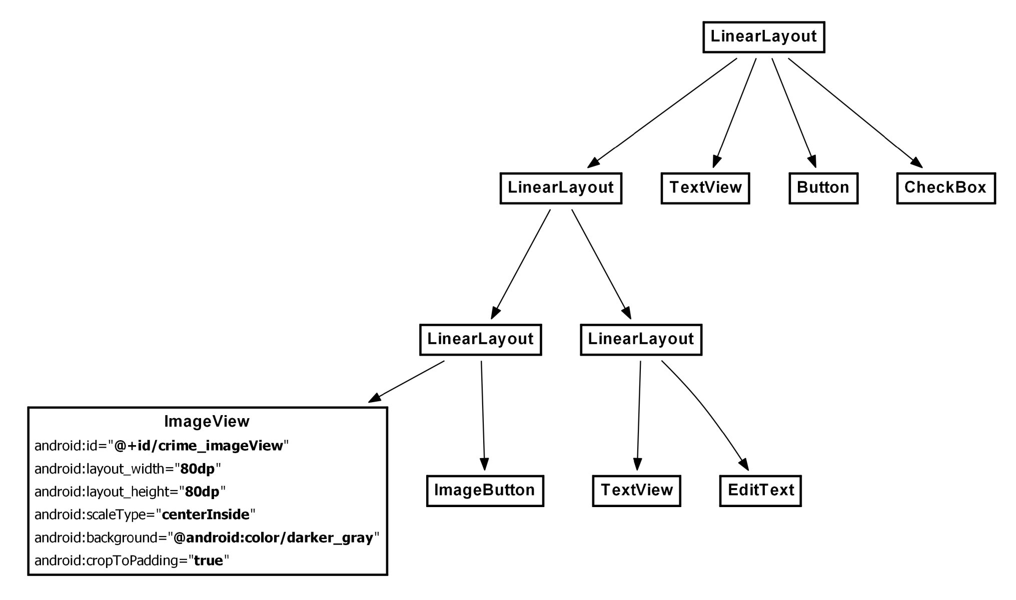


Рис. 20.8. Макет CrimeFragment с ImageView (layout/fragment_crime.xml)

Также нам понадобится виджет `ImageView` в альбомном макете (рис. 20.9).

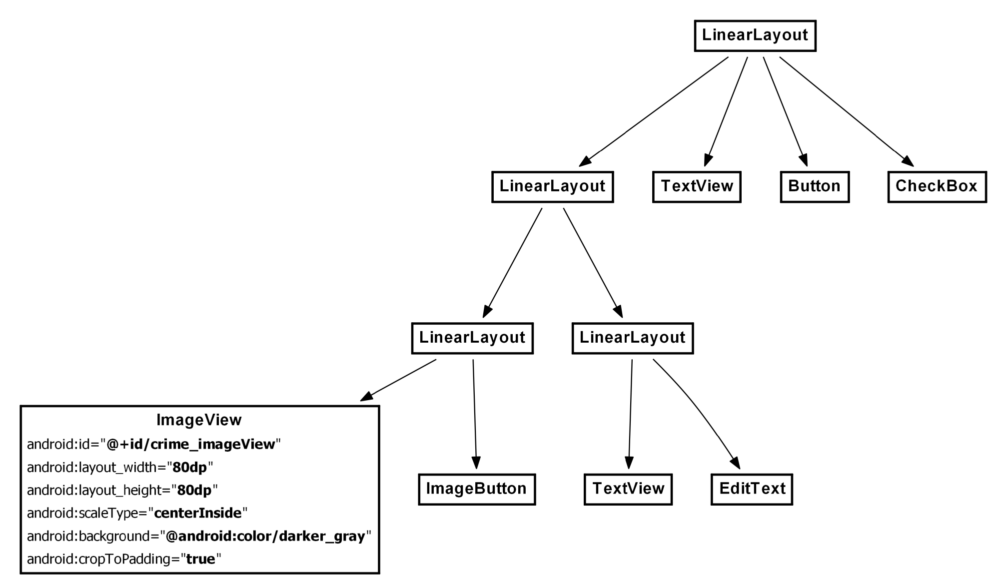


Рис. 20.9. Альбомный макет с ImageView (layout-land/fragment_crime.xml)

В файле CrimeFragment.java определите новое поле и получите ссылку на `ImageView` в методе `onCreateView(…)`.

Листинг 20.11. Подготовка ImageView (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private ImageButton mPhotoButton;
    private ImageView mPhotoView;

    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Launch the camera activity
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivityForResult(i, REQUEST_PHOTO);
            }
        });

        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);
        ...
    }
```

Чтобы убедиться в том, что виджет `ImageView` находится на положенном месте, просмотрите макет или запустите приложение CriminalIntent.

Обработка изображения

Для вывода изображения на виджете ImageView потребуется некоторая предварительная обработка, потому что файлы, полученные с камеры, могут быть просто огромными. С каждым годом фирмы-производители устанавливают на своих телефонах все большие и современные камеры. Технический прогресс радует пользователей, но создает проблемы программистам.

На момент написания книги самые современные телефоны на платформе Android оснащались 8-мегапиксельными камерами. Изображения такого размера мгновенно израсходуют всю доступную память, поэтому нам понадобится код, который будет масштабировать изображение перед загрузкой, а также код стирания ненужных изображений.

Добавление масштабированных фотографий в ImageView

В пакете com.bignerdranch.android.criminalintent создайте новый класс с именем PictureUtils. Добавьте в файл PictureUtils.java метод, масштабирующий изображение по размеру стандартного экрана устройства.

Листинг 20.12. Добавление класса PictureUtils (PictureUtils.java)

```
public class PictureUtils {
    /**
     * Получение объекта BitmapDrawable по данным локального файла,
     * масштабированного по текущим размерам окна.
     */
    @SuppressWarnings("deprecation")
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {
        Display display = a.getWindowManager().getDefaultDisplay();
        float destWidth = display.getWidth();
        float destHeight = display.getHeight();

        // Чтение размеров изображения на диске
        BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        BitmapFactory.decodeFile(path, options);

        float srcWidth = options.outWidth;
        float srcHeight = options.outHeight;

        int inSampleSize = 1;
        if (srcHeight > destHeight || srcWidth > destWidth) {
            if (srcWidth > srcHeight) {
                inSampleSize = Math.round(srcHeight / destHeight);
            } else {
                inSampleSize = Math.round(srcWidth / destWidth);
            }
        }

        options = new BitmapFactory.Options();
        options.inSampleSize = inSampleSize;
```

продолжение ⇗

Листинг 20.12 (*продолжение*)

```
        Bitmap bitmap = BitmapFactory.decodeFile(path, options);
        return new BitmapDrawable(a.getResources(), bitmap);
    }
}
```

Методы Display.getWidth() и Display.getHeight() считаются устаревшими (deprecated). Эта тема более подробно рассматривается в конце главы.

В идеале изображение стоило бы масштабировать так, чтобы оно точно соответствовало размерам ImageView. Однако размер представления, в котором выводится представление, часто бывает недоступен в нужный момент. Например, в методе onCreateView(…) мы не можем получить размер ImageView. Для надежности изображение масштабируется по размерам текущего экрана устройства, который доступен всегда. Представление, в котором будет выводиться фотография, может быть меньше стандартного размера экрана, но больше быть не может.

Затем добавьте в CrimeFragment закрытый метод, который связывает масштабированную версию изображения с ImageView.

Листинг 20.13. Добавление showPhoto() (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    ...
}

private void showPhoto() {
    // Назначение изображения, полученного на основе фотографии
    Photo p = mCrime.getPhoto();
    BitmapDrawable b = null;
    if (p != null) {
        String path = getActivity()
            .getFileStreamPath(p.getFilename()).getAbsolutePath();
        b = PictureUtils.getScaledDrawable(getActivity(), path);
    }
    mPhotoView.setImageDrawable(b);
}
```

Включите в файл CrimeFragment.java реализацию onStart() с вызовом showPhoto(), чтобы фотография была готова к тому моменту, как пользователь увидит CrimeFragment.

Листинг 20.14. Загрузка изображения (CrimeFragment.java)

```
...
private void showPhoto() {
    // Назначение изображения, полученного на основе фотографии
    Photo p = mCrime.getPhoto();
    BitmapDrawable b = null;
    if (p != null) {
        String path = getActivity()
            .getFileStreamPath(p.getFilename()).getAbsolutePath();
        b = PictureUtils.getScaledDrawable(getActivity(), path);
    }
```

```
    mPhotoButton.setImageDrawable(b);
}

@Override
public void onStart() {
    super.onStart();
    showPhoto();
}
```

Включите в метод `CrimeFragment.onActivityResult(…)` вызов `showPhoto()`, чтобы изображение было видимым при возвращении пользователя из `CrimeCameraActivity`.

Листинг 20.15. Вызов showPhoto() в onActivityResult(…) (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_PHOTO) {
        // Создание нового объекта Photo и связывание его с Crime
        String filename = data
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename != null) {
            Photo p = new Photo(filename);
            mCrime.setPhoto(p);
            showPhoto();
            Log.i(TAG, "Crime: " + mCrime.getTitle() + "has a photo");
        }
    }
}
```

Выгрузка изображения

Включите в класс `PictureUtils` метод для уничтожения экземпляра `BitmapDrawable`, связанного с `ImageView` (если он существует).

Листинг 20.16. Очистка данных (PictureUtils.java)

```
public class PictureUtils {
    /**
     * ...
     */
    @SuppressWarnings("deprecation")
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {
        ...
    }

    public static void cleanImageView(ImageView imageView) {
        if (!(imageView.getDrawable() instanceof BitmapDrawable))
            return;

        // Стирание изображения для экономии памяти
        BitmapDrawable b = (BitmapDrawable)imageView.getDrawable();
        b.getBitmap().recycle();
        imageView.setImageDrawable(null);
    }
}
```

О вызове `Bitmap.recycle()` стоит рассказать подробнее. В документации говорится, что вызывать этот метод не обязательно, но это не так. `Bitmap.recycle()` освобождает системную (native) память, занимаемую растровым изображением. Это большая часть содержания объекта растрового изображения. (Системная память может содержать больший или меньший объем данных в зависимости от версии Android. До Honeycomb в ней хранились все данные объектов Java `Bitmap`.)

Если не освободить память явным вызовом `recycle()`, она в конечном итоге все равно будет освобождена. Однако освобождение произойдет когда-то в будущем в *завершителе* (finalizer), а не при уничтожении самого изображения в ходе уборки мусора. Соответственно возникает вероятность исчерпания свободной памяти до вызова завершителя.

Момент выполнения завершителя заранее неизвестен, поэтому такие ошибки сложно отслеживать и воспроизводить. Таким образом, при большом размере изображений (как в нашем случае) лучше вызвать `recycle()` для предотвращения неприятных ошибок памяти.

Включите в `CrimeFragment` реализацию `onStop()` с вызовом `cleanImageView(…)`.

Листинг 20.17. Выгрузка изображения (CrimeFragment.java)

```
@Override
public void onStart() {
    super.onStart();
    showPhoto();
}

@Override
public void onStop() {
    super.onStop();
    PictureUtils.cleanImageView(mPhotoView);
}
```

Загрузка изображений в `onStart()` с выгрузкой в `onStop()` — полезная практика. Эти методы отмечают точки, в которых активность может быть видна пользователю. Если же выполнять загрузку и выгрузку в `onResume()` и `onPause()`, результат может оказаться неожиданным для пользователя.

Приостановленная активность все равно может быть частично видимой — если, например, поверх нее открывается активность, не занимающая весь экран. Если использовать `onResume()` и `onPause()`, изображения в таких ситуациях будут исчезать и появляться. Лучше загружать изображения, как только ваша активность становится доступной, и откладывать выгрузку до того момента, когда активность гарантированно не видна на экране.

Запустите приложение CriminalIntent. Сделайте снимок и убедитесь в том, что он появился в `ImageView`. Закройте CriminalIntent и запустите приложение заново. Убедитесь в том, что при возвращении к тому же элементу списка преступлений фотография отображается, как и положено.

Ориентация `CrimeCameraActivity` подсказывает пользователю, что фотографии следует делать в альбомной ориентации. Если фотография будет сделана в книжной ориентации, то изображение может быть неправильно расположено на кнопке.

Этот недостаток будет исправлен в первом упражнении.

Отображение полноразмерных изображений в DialogFragment

Наша работа над поддержкой камеры завершится отображением увеличенных версий фотографий `Crime`.

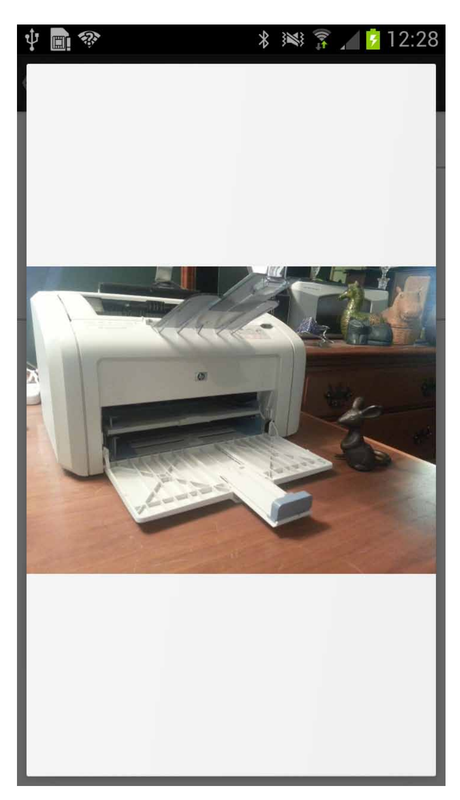


Рис. 20.10. DialogFragment с увеличенным изображением

Создайте новый класс в пакете `com.bignerdranch.android.criminalintent`. Присвойте классу имя `ImageFragment`; назначьте его субклассом `DialogFragment`.

В аргументах фрагмента `ImageFragment` должен передаваться путь к файлу фотографии для `Crime`. В файле ImageFragment.java добавьте метод `newInstance(String)`, который получает путь к файлу и помещает его в пакет аргументов, как показано в листинге 20.18.

Листинг 20.18. Создание ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        Bundle args = new Bundle();
```

продолжение ⇗

Листинг 20.18 (*продолжение*)

```
        args.putSerializable(EXTRA_IMAGE_PATH, imagePath);

        ImageFragment fragment = new ImageFragment();
        fragment.setArguments(args);
        fragment.setStyle(DialogFragment.STYLE_NO_TITLE, 0);

        return fragment;
    }
}
```

Фрагменту назначается стиль `DialogFragment.STYLE_NO_TITLE`, с которым фрагмент имеет минималистское оформление, показанное на рис. 20.10.

`ImageFragment` не нужны заголовок и кнопки, предоставляемые `AlertDialog`. Если ваш фрагмент может обойтись без них, лучше использовать более элегантное, быстрое и гибкое решение с переопределением `onCreateView(…)` и использованием простого объекта `View` (вместо переопределения `onCreateDialog(…)` и использования Dialog).

В файле ImageFragment.java переопределите метод `onCreateView(…)`, чтобы он создавал объект `ImageView` «с нуля» с получением пути к файлу из своих аргументов. Затем получите масштабированную версию изображения и свяжите ее с `ImageView`.

Также переопределите `onDestroyView()`, который должен освобождать память, когда надобность в изображении отпадает.

Листинг 20.19. Создание ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        ...
    }

    private ImageView mImageView;

    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup parent, Bundle savedInstanceState) {
        mImageView = new ImageView(getActivity());
        String path = (String)getArguments().getSerializable(EXTRA_IMAGE_PATH);
        BitmapDrawable image = PictureUtils.getScaledDrawable(getActivity(), path);

        mImageView.setImageDrawable(image);

        return mImageView;
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        PictureUtils.cleanImageView(mImageView);
    }
}
```

Наконец, необходимо вывести это диалоговое окно из `CrimeFragment`. В файле CrimeFragment.java добавьте слушателя для `mPhotoView`. В его реализации создайте экземпляр `ImageFragment` и добавьте его в экземпляр `FragmentManager` активности `CrimePagerActivity` вызовом метода `show(…)` для `ImageFragment`. Также понадобится строковая константа для идентификации `ImageFragment` в `FragmentManager`.

Листинг 20.20. Отображение ImageFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private static final String DIALOG_IMAGE = "image";

    ...

    @Override
    @TargetApi(11)
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        ...

        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);
        mPhotoView.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Photo p = mCrime.getPhoto();
                if (p == null)
                    return;

                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                String path = getActivity()
                    .getFileStreamPath(p.getFilename()).getAbsolutePath();
                ImageFragment.newInstance(path)
                    .show(fm, DIALOG_IMAGE);
            }
        });
        ...
    }
```

Запустите приложение CriminalIntent. Сделайте снимок и убедитесь в том, что теперь приложение позволяет просмотреть фотографию сцены преступления во всех подробностях.

Упражнение. Ориентация изображения в Crime

Иногда пользователь хочет сделать снимок в книжной ориентации. Найдите в документации API информацию о том, как обнаружить текущую ориентацию. Сохраните правильную ориентацию в `Photo` и используйте ее для поворота изображения в `CrimeFragment` и `ImageFragment`.

Упражнение. Удаление фотографий

В текущей версии приложения можно заменить фотографию `Crime`, но при этом старый файл остается и занимает место на диске. Добавьте в метод `onActivityResult(int,`

int, Intent) класса CrimeFragment код проверки существующей фотографии и удаления соответствующего файла с диска.

В качестве дополнительного задания предоставьте пользователю возможность удаления фотографии без замены. Реализуйте в CrimeFragment контекстное меню и/или режим контекстных действий, активизируемый долгим нажатием на миниатюре изображения. Команда меню Delete Photo удаляет фотографию с диска, из модели и ImageView.

Для любознательных: устаревшие конструкции в Android

В главе 19, при назначении размера области предварительного просмотра камеры мы использовали устаревший метод и устаревшую константу. В этой главе тоже использовались устаревшие методы. У наблюдательного читателя может возникнуть вопрос: «Что это вообще значит?»

Начнем с того, что же понимать под самим термином. Если некоторая часть API считается устаревшей, это означает, что она более не является необходимой. Иногда устаревание происходит из-за того, что выполняемая операция стала ненужной — как в случае с методом SurfaceHolder.setType(int) и константой SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS, использованными в главе 19.

В старых версиях Android экземпляр SurfaceHolder должен был настраиваться в соответствии с предполагаемым использованием. Сейчас это уже не нужно, поэтому метод setType(…) стал бесполезным.

В других случаях это означает, что метод был заменен новым методом, который по какой-то причине является предпочтительным. Например, класс BitmapDrawable содержит устаревший конструктор BitmapDrawable(Bitmap), при использовании которого часто допускались ошибки. А может быть, новый метод «чище» с точки зрения архитектуры, как, например, метод View.setBackgroundDrawable(Drawable). Также можно вспомнить методы Display.getWidth() и Display.getHeight(), которые использовались ранее. Сейчас они заменены одним методом getSize(Point), который предотвращает ошибки, возникавшие при последовательном вызове getWidth() и getHeight().

Проблема устаревания кода решается по-разному в зависимости от того, на какой платформе вы работаете. Две крайности воплощены в двух подходах, которые вам, вероятно, знакомы — подходе Microsoft и подходе Apple.

В подходе Microsoft части API устаревают, но никогда не удаляются. Это связано с тем, что в мировоззрении Microsoft важнейшим фактором является работоспособность наибольшего количества программ в любой версии ОС. Когда Microsoft представляет общедоступный API, он всегда поддерживается. Порой доходит до сохранения ошибочного, недокументированного поведения для обеспечения обратной совместимости. Из-за этого мир Microsoft порой выглядит довольно странно.

С другой стороны, в мире Apple API удаляются из ОС вскоре после того, как они были объявлены устаревшими. Руководители мира Apple стремятся иметь чистую,

красивую ОС. Их не волнует, сколько API придется убить для достижения этой цели. В результате мир Apple очень чист и аккуратен, но старые программы часто перестают работать без активного обновления.

В мире Apple для поддержки как старых, так и новых операционных систем приходится использовать конструкции следующего вида:

```
float destWidth;
float destHeight;

if (Build.VERSION.SDK_INT > Build.VERSION_CODES.HONEYCOMB_MR2) {
    Point size = display.getSize();
    destWidth = size.x;
    destHeight = size.y;
} else {
    destWidth = display.getWidth();
    destHeight = display.getHeight();
}
```

Дело в том, что в мире Apple методы `getWidth()` и `getHeight()`, вероятно, скоро исчезнут. Нужно действовать осторожно, чтобы случайно не вызвать несуществующий метод.

Нельзя сказать. что идеология Android совпадает с подходом Microsoft, но по крайней мере достаточно близка к нему. Каждая версия Android SDK в основном сохраняет обратную совместимость с предыдущей версией, а это означает, что методы API почти никогда не удаляются. Значит, вам не нужно избегать вызова старых методов. В книге мы будем использовать устаревшие методы там, где это необходимо, и будем подавлять предупреждения при помощи аннотаций. И все же старайтесь не использовать устаревшие API без необходимости, чтобы ваш код оставался стильным и элегантным.

21 Неявные интенты

В Android можно запустить активность из другого приложения на устройстве при помощи *неявного интента* (implicit intent). В явном интенте задается класс запускаемой активности, а ОС запускает его. В неявном интенте вы описываете операцию, которую необходимо выполнить, а ОС запускает активность соответствующего приложения.

В приложении CriminalIntent мы будем использовать неявные интенты для выбора подозреваемых из списка контактов пользователя и отправки текстовых отчетов о преступлении. Пользователь выбирает подозреваемого в контактном приложении, установленном на устройстве, и получает список приложений для отправки отчета.

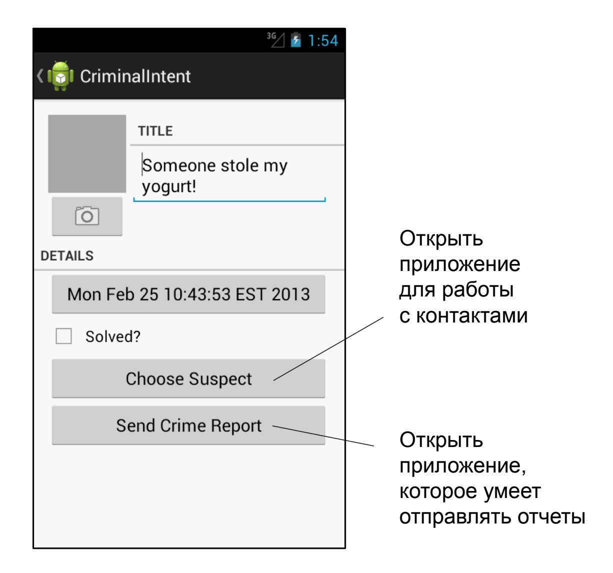


Рис. 21.1. Открытие приложений для выбора контактов и отправки отчетов

Использовать функциональность других приложений при помощи неявных интентов намного проще, чем писать собственные реализации стандартных задач. Пользователям также нравится работать с приложениями, которые им уже хорошо знакомы, в сочетании с вашим приложением.

Прежде чем создавать неявные интенты, необходимо выполнить с CriminalIntent ряд подготовительных действий:

- добавить в макеты `CrimeFragment` кнопки выбора подозреваемого и отправки отчета;
- добавить в класс `Crime` поле `mSuspect`, в котором будет храниться имя подозреваемого;
- создать отчет о преступлении с использованием форматных строк ресурсов.

Добавление кнопок

Начнем с включения в макеты `CrimeFragment` новых кнопок. Прежде всего добавьте строки, которые будут отображаться на кнопках.

Листинг 21.1. Добавление строк для надписей на кнопках (strings.xml)

```
  <string name="take">Take!</string>
  <string name="crime_suspect_text">Choose Suspect</string>
  <string name="crime_report_text">Send Crime Report</string>
</resources>
```

Добавьте в файл layout/fragment_crime.xml два виджета `Button`, представленных на рис. 21.2. Обратите внимание: на диаграмме не показан первый виджет `LinearLayout`, чтобы вы могли сосредоточиться на новых и интересных частях диаграммы.

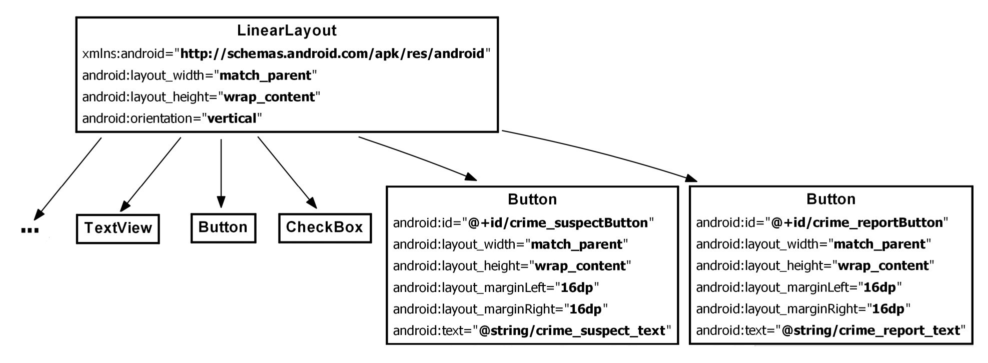


Рис. 21.2. Добавление кнопок для выбора контактов и отправки отчетов (layout/fragment_crime.xml)

В альбомном макете мы назначим новые кнопки потомками нового горизонтального виджета `LinearLayout`, расположенного под виджетом с кнопкой даты и флажком.

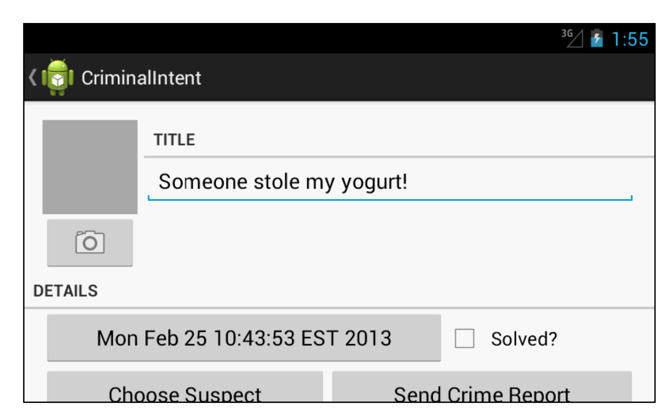


Рис. 21.3. Нижние кнопки частично скрыты в альбомном режиме

Как видно из рис. 21.3, на малых экранах новые кнопки видны лишь частично. Чтобы решить эту проблему, мы разместим весь макет для альбомного режима в `ScrollView`. Новый макет представлен на рис. 21.4. Так как корневым элементом теперь является `ScrollView`, не забудьте переместить пространство имен из предыдущего корневого элемента в `ScrollView`.

ScrollView
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:layout_width="**match_parent**"
android:layout_height="**match_parent**"

LinearLayout
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"
android:orientation="**vertical**"

...

TextView

...

LinearLayout
android:layout_width="**match_parent**"
android:layout_height="**wrap_content**"
android:layout_marginLeft="**16dp**"
android:layout_marginRight="**16dp**"
android:orientation="**horizontal**"

Button
android:id="**@+id/crime_suspectButton**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:layout_weight="**1**"
android:text="**@string/crime_suspect_text**"

Button
android:id="**@+id/crime_reportButton**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:layout_weight="**1**"
android:text="**@string/crime_report_text**"

Рис. 21.4. Добавление кнопок для выбора контактов и отправки отчетов (layout_land/fragment_crime.xml)

На этой стадии вы можете проверить макеты в области предварительного просмотра или запустить приложение CriminalIntent, чтобы убедиться в правильности расположения новых кнопок.

Добавление имени в уровень модели

Вернитесь к файлу Crime.java, добавьте новую константу JSON и поле для хранения имени подозреваемого. Также измените методы JSON для выполнения сериализации/десериализации из кода JSON и добавьте новые методы доступа.

Листинг 21.2. Добавление поля для имени подозреваемого (Crime.java)

```
public class Crime {
    ...
    private static final String JSON_PHOTO = "photo";
    private static final String JSON_SUSPECT = "suspect";

    ...
    private Photo mPhoto;
    private String mSuspect;

    public Crime(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        ...
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
        if (json.has(JSON_SUSPECT))
            mSuspect = json.getString(JSON_SUSPECT);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
            json.put(JSON_SUSPECT, mSuspect);
        return json;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }

    public String getSuspect() {
        return mSuspect;
    }
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

Форматные строки

Последним подготовительным шагом станет создание шаблона отчета о преступлении, который заполняется информацией о конкретном преступлении. Так как

подробная информация недоступна до стадии выполнения, необходимо использовать форматную строку с заполнителями, которые будут заменяться во время выполнения. Форматная строка будет выглядеть так:

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and %4$s
```

Поля `%1$s`, `%2$s` и т. д. — заполнители для строковых аргументов. В коде вы вызываете `getString(…)` и передаете форматную строку и еще четыре строки в том порядке, в каком они должны заменять заполнители.

Сначала добавьте в strings.xml строки из листинга 21.3.

Листинг 21.3. Добавление строковых ресурсов (strings.xml)

```
  <string name="crime_suspect_text">Choose Suspect</string>
  <string name="crime_report_text">Send Crime Report</string>
  <string name="crime_report">%1$s!
    The crime was discovered on %2$s. %3$s, and %4$s
  </string>
  <string name="crime_report_solved">The case is solved</string>
  <string name="crime_report_unsolved">The case is not solved</string>
  <string name="crime_report_no_suspect">There is no suspect.</string>
  <string name="crime_report_suspect">The suspect is %s.</string>
  <string name="crime_report_subject">CriminalIntent Crime Report</string>
  <string name="send_report">Send crime report via</string>

</resources>
```

В файле CrimeFragment.java добавьте метод, который создает четыре строки, соединяет их и возвращает полный отчет.

Листинг 21.4. Добавление метода getCrimeReport() (CrimeFragment.java)

```
private String getCrimeReport() {
    String solvedString = null;
    if (mCrime.isSolved()) {
        solvedString = getString(R.string.crime_report_solved);
    } else {
        solvedString = getString(R.string.crime_report_unsolved);
    }

    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat.format(dateFormat, mCrime.getDate()).toString();

    String suspect = mCrime.getSuspect();
    if (suspect == null) {
        suspect = getString(R.string.crime_report_no_suspect);
    } else {
        suspect = getString(R.string.crime_report_suspect, suspect);
    }
    String report = getString(R.string.crime_report,
        mCrime.getTitle(), dateString, solvedString, suspect);
    return report;
}
```

Приготовления завершены, теперь можно непосредственно заняться неявными интентами.

Использование неявных интентов

Объект Intent описывает для ОС некую операцию, которую вы хотите выполнить. Для явных интентов, использовавшихся до настоящего момента, разработчик явно указывает активность, которую должна запустить ОС.

```
Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
startActivity(i);
```

Для неявных интентов разработчик описывает выполняемую операцию, а ОС запускает активность, которая ранее сообщила о том, что она способна выполнять эту операцию. Если ОС находит несколько таких активностей, пользователю предлагается выбрать нужную.

Компоненты неявного интента

Ниже перечислены важнейшие составляющие интента, используемые для определения выполняемой операции.

Выполняемое *действие* (action) — обычно определяется константами из класса Intent. Так, для просмотра URL-адреса используется константа Intent.ACTION_VIEW, а для отправки данных — константа Intent.ACTION_SEND.

Местонахождение *данных* — это может быть как ссылка на данные, находящиеся за пределами устройства (скажем, URL веб-страницы), так и URI файла или URI контента, ссылающийся на запись ContentProvider.

Тип данных, с которыми работает действие, — тип MIME (например, text/html или audio/mpeg3). Если в интент включено местонахождение данных, то тип обычно удается определить по этим данным.

Необязательные *категории* — если действие используется для описания выполняемой операции, категория обычно описывает, где, когда или как вы пытаетесь использовать операцию. Android использует категорию android.intent.category.LAUNCHER для обозначения активностей, которые должны отображаться в лаунчере приложений верхнего уровня. С другой стороны, категория android.intent.category.INFO обозначает активность, которая выдает пользователю информацию о пакете, но не отображается в лаунчере.

Простой неявный интент для просмотра веб-сайта включает действие Intent.ACTION_VIEW и объект данных Uri с URL-адресом сайта.

На основании этой информации ОС запускает соответствующую активность соответствующего приложения. (Если ОС обнаруживает более одного кандидата, пользователю предлагается принять решение.)

Активность сообщает о себе как об исполнителе для ACTION_VIEW при помощи фильтра интентов в манифесте. Например, если вы пишете приложение-браузер, вы включаете следующий фильтр интентов в объявление активности, реагирующей на ACTION_VIEW.

```
<activity
  android:name=".BrowserActivity"
  android:label="@string/app_name" >
```

продолжение ⇗

```
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" android:host="www.bignerdranch.com" />
  </intent-filter>
</activity>
```

Категория `DEFAULT` должна явно задаваться в фильтрах интентов. Элемент `action` в фильтре интентов сообщает ОС, что активность способна выполнять операцию, а категория `DEFAULT` — что она желает рассматриваться среди кандидатов на выполнение операции. Категория `DEFAULT` неявно добавляется к почти любому неявному интенту. (Единственное исключение составляет категория `LAUNCHER`, с которой мы будем работать в главе 23.)

Неявные интенты, как и явные, также могут включать дополнения. Однако дополнения неявного интента не используются ОС для поиска соответствующей активности.

Также следует отметить, что компоненты действия и данных интента могут использоваться в сочетании с явными интентами. Результат эквивалентен тому, что вы приказываете конкретной активности выполнить конкретную операцию.

Отправка отчета

Чтобы увидеть на практике, как работает эта схема, мы создадим неявный интент для отправки отчета о преступлении в приложении CriminalIntent. Операция, которую нужно выполнить, — отправка простого текста; отчет представляет собой строку. Таким образом, действие неявного интента будет представлено константой `ACTION_SEND`. Интент не содержит ссылок на данные и не имеет категорий, но определяет тип `text/plain`.

В методе `CrimeFragment.onCreateView(…)` получите ссылку на кнопку Send Crime Report и назначьте для нее слушателя. В реализации слушателя создайте неявный интент и передайте его `startActivity(Intent)`.

Листинг 21.5. Отправка отчета о преступлении (CrimeFragment.java)

```
    ...
    Button reportButton = (Button)v.findViewById(R.id.crime_reportButton);
    reportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
            i.putExtra(Intent.EXTRA_SUBJECT,
                getString(R.string.crime_report_subject));
            startActivity(i);
        }
    });

    return v;
}
```

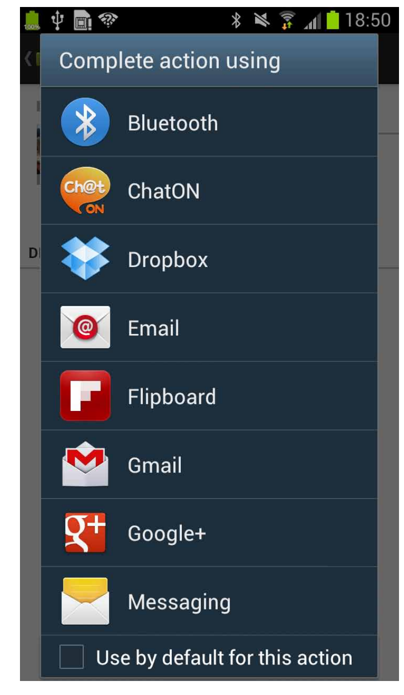


Рис. 21.5. Активности, заявившие о своей готовности выполнить отправку отчета

Здесь мы используем конструктор `Intent`, который получает строку с константой, описывающей действие. Также существуют другие конструкторы, которые могут использоваться в зависимости от вида создаваемого неявного интента. Информацию о них можно найти в справочной документации `Intent`. Конструктора, получающего тип, не существует, поэтому мы задаем его явно.

Текст отчета и строка темы включаются в дополнения. Обратите внимание на использование в них констант, определенных в классе `Intent`. Любая активность, реагирующая на интент, знает эти константы и то, что следует делать с ассоциированными значениями.

Запустите приложение CriminalIntent и нажмите кнопку Send Crime Report. Так как этот интент с большой вероятностью совпадет со многими активностями на устройстве, скорее всего, на экране появится список активностей:

Если на экране появился список, выберите нужный вариант. Вы увидите, что отчет о преступлении загружается в выбранном вами приложении. Вам остается лишь ввести адрес и отправить его.

Если список не появился, это может означать одно из двух: либо вы уже назначили приложение по умолчанию для идентичного неявного интента, либо на вашем устройстве имеется всего одна активность, способная реагировать на этот интент.

Часто лучшим вариантом оказывается использование приложения по умолчанию, выбранного пользователем для данного действия. Впрочем, в приложении CriminalIntent лучше всегда предоставлять пользователю выбор: сегодня пользователь предпочтет не поднимать шума и отправит отчет по электронной почте, а завтра выставит нарушителя на общественный позор в Твиттере.

Вы можете создать список, который будет отображаться каждый раз при использовании неявного интента для запуска активности. После создания неявного интента способом, показанным ранее, вы вызываете следующий метод `Intent` и передаете ему неявный интент и строку с заголовком:

```
public static Intent createChooser(Intent target, String title)
```

Затем интент, возвращенный `createChooser(…)`, передается `startActivity(…)`.

В файле CrimeFragment.java создайте список выбора для отображения активностей, реагирующих на неявный интент.

Листинг 21.6. Использование списка выбора (CrimeFragment.java)

```
public void onClick(View v) {
    Intent i = new Intent(Intent.ACTION_SEND);
    i.setType("text/plain");
    i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
    i.putExtra(Intent.EXTRA_SUBJECT,
        getString(R.string.crime_report_subject));
    i = Intent.createChooser(i, getString(R.string.send_report));
    startActivity(i);
}
```

Запустите приложение CriminalIntent и нажмите кнопку Send Crime Report. Если в системе имеется несколько активностей, способных обработать ваш интент, на экране появляется список для выбора.

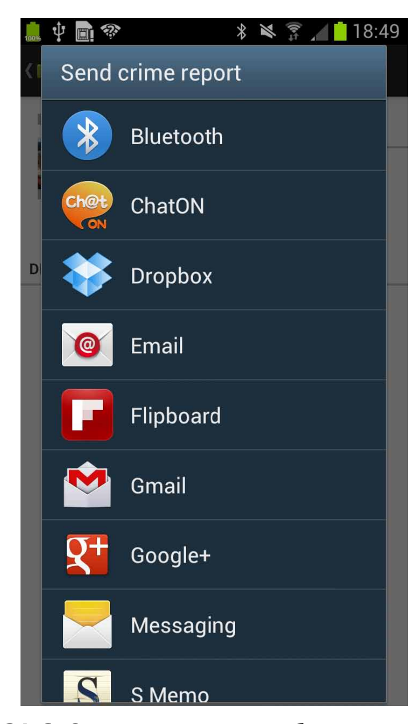


Рис. 21.6. Отправка текста с выбором активности

Запрос контакта у Android

Теперь мы создадим другой неявный интент, который предлагает пользователю выбрать подозреваемого из списка контактов. Для этого неявного интента будет определено действие и местонахождение соответствующих данных. Действие задается константой `Intent.ACTION_PICK`, а местонахождение данных — `ContactsContract.`

Contacts.CONTENT_URI. Короче говоря, вы просите Android помочь с выбором записи из базы данных контактов.

Запущенная активность должна вернуть результат, поэтому мы передаем интент через startActivityForResult(…) вместе с кодом запроса. Добавьте в файл CrimeFragment.java константу для кода запроса и поле для кнопки.

Листинг 21.7. Добавление поля для кнопки подозреваемого (CrimeFragment.java)

```
...
private static final int REQUEST_PHOTO = 1;
private static final int REQUEST_CONTACT = 2;
...

private ImageButton mPhotoButton;
private Button mSuspectButton;
...
```

В конце onCreateView(…) получите ссылку на кнопку и назначьте ей слушателя. В реализации слушателя создайте неявный интент и передайте его startActivityForResult(…). Также выведите на кнопке имя подозреваемого (если оно содержится в Crime).

Листинг 21.8. Отправка неявного интента (CrimeFragment.java)

```
    ...

    mSuspectButton = (Button)v.findViewById(R.id.crime_suspectButton);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_PICK,
                 ContactsContract.Contacts.CONTENT_URI);
            startActivityForResult(i, REQUEST_CONTACT);
        }
    });

    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    return v;
}
```

Запустите приложение CriminalIntent и нажмите кнопку Choose Suspect. На экране появляется список контактов.

Если у вас установлено другое контактное приложение, экран будет выглядеть иначе. Это еще одно преимущество неявных интентов: вам не нужно знать название контактного приложения, чтобы использовать его из своего приложения. Соответственно пользователь может установить то приложение, которое считает нужным, а ОС найдет и запустит его.

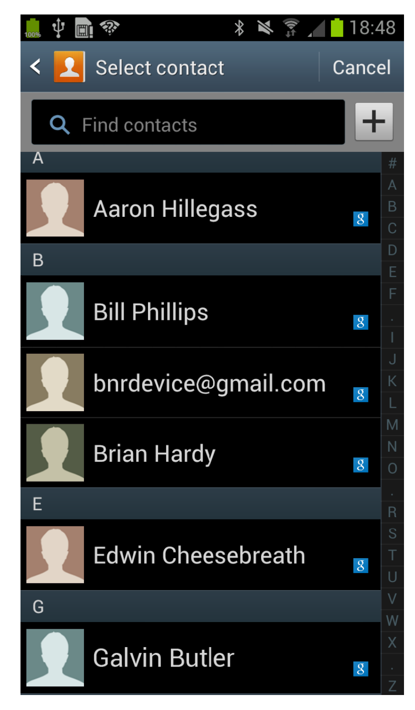


Рис. 21.7. Список подозреваемых

Получение данных из списка контактов

Теперь необходимо получить результат от контактного приложения. Контактная информация совместно используется многими приложениями, поэтому Android предоставляет расширенный API для работы с контактными данными через `ContentProvider`. Экземпляры этого класса инкапсулируют базы данных и предоставляют доступ к ним другим приложениям. Обращение к `ContentProvider` осуществляется через `ContentResolver`.

Так как активность запускалась с возвращением результата с использованием `ACTION_PICK`, вы можете получить интент вызовом `onActivityResult(…)`. Интент включает URI данных — ссылку на конкретный контакт, выбранный пользователем.

В файле CrimeFragment.java добавьте следующий код в реализацию `onActivityResult(…)` из `CrimeFragment`.

Листинг 21.9. Получение имени контакта (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
```

```
        ...
    } else if (requestCode == REQUEST_CONTACT) {
        Uri contactUri = data.getData();

        // Определение полей, значения которых должны быть
        // возвращены запросом.
        String[] queryFields = new String[] {
            ContactsContract.Contacts.DISPLAY_NAME
        };
        // Выполнение запроса - contactUri здесь выполняет функции
        // условия "where"
        Cursor c = getActivity().getContentResolver()
            .query(contactUri, queryFields, null, null, null);

        // Проверка получения результатов
        if (c.getCount() == 0) {
            c.close();
            return;
        }

        // Извлечение первого столбца данных - имени подозреваемого.
        c.moveToFirst();
        String suspect = c.getString(0);
        mCrime.setSuspect(suspect);
        mSuspectButton.setText(suspect);
        c.close();
    }
}
```

В листинге 21.9 создается запрос, который запрашивает все отображаемые имена контактов из возвращаемых данных. Затем мы выдаем запрос к базе данных контактов и получаем объект `Cursor` для работы с ней. Так как мы знаем, что курсор содержит всего один элемент, мы переходим к первому элементу и используем его как строку. Эта строка содержит имя подозреваемого, которое мы используем для задания подозреваемого в `Crime` и текста кнопки Choose Suspect.

(База данных контактов сама по себе является достаточно обширной темой. Здесь она не рассматривается. Если вам захочется узнать больше, обратитесь к руководству по Contacts Provider API: *http://developer.android.com/guide/topics/providers/contacts-provider.html*.)

Разрешения контактов

Как получить разрешение на чтение из базы данных контактов? Контактное приложение распространяет свои разрешения на вас. Оно обладает полными разрешениями на обращение к базе данных. Когда контактное приложение возвращает родительской активности URI данных в интенте, оно также добавляет флаг `Intent.FLAG_GRANT_READ_URI_PERMISSION`. Этот флаг сообщает Android, что родительской активности в CriminalIntent следует разрешить однократное использование этих данных. Такой подход работает хорошо, потому что фактически нам нужен доступ не ко всей базе данных контактов, а к одному контакту в этой базе.

Проверка реагирующих активностей

На два неявных интента, созданных в этой главе, кто-то гарантированно отреагирует. На устройстве Android заведомо присутствует то или иное приложение для работы с электронной почтой и контактное приложение. А если вы создаете другой неявный интент для устройства, на котором может не оказаться подходящих активностей? Если ОС не найдет подходящую активность, в приложении происходит сбой.

Проблема решается предварительной проверкой того, от какой части ОС поступил вызов PackageManager. Код выглядит примерно так:

```
PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(yourIntent, 0);
boolean isIntentSafe = activities.size() > 0;
```

Интент передается методу queryIntentActivities(…) класса PackageManager. Метод возвращает список объектов, содержащих метаданные об активностях, отреагировавших на переданный интент. Остается убедиться в том, что список содержит хотя бы один элемент — то есть на устройстве имеется хотя бы одна активность, реагирующая на интент.

Выполнение этой проверки в onCreateView(…) позволяет отключить варианты, на которые устройство не сможет отреагировать.

Упражнение. Другой неявный интент

Возможно, вместо отправки отчета разгневанный пользователь предпочтет разобраться с подозреваемым по телефону. Добавим новую кнопку для звонка указанному подозреваемому.

Нам понадобится извлечь из базы данных контактов телефонный номер, после чего можно создать неявный интент с URI телефона:

```
Uri number = Uri.parse("tel:5551234");
```

При этом может использоваться действие Intent.ACTION_DIAL или Intent.ACTION_CALL. ACTION_CALL запускает телефонное приложение и немедленно осуществляет звонок по номеру, отправленному в интенте; ACTION_DIAL только вводит номер и ждет, пока пользователь инициирует звонок.

Мы рекомендуем использовать ACTION_DIAL. Режим ACTION_CALL может быть ограничен, и для него определенно потребуются разрешения. Кроме того, у пользователя будет возможность немного остыть перед нажатием кнопки вызова.

22 Двухпанельные интерфейсы

В этой главе мы создадим для CriminalIntent планшетный интерфейс, в котором пользователь может одновременно видеть и взаимодействовать со списком преступлений и подробным описанием конкретного преступления. На рис. 22.1 изображен такой интерфейс типа «список-детализация».

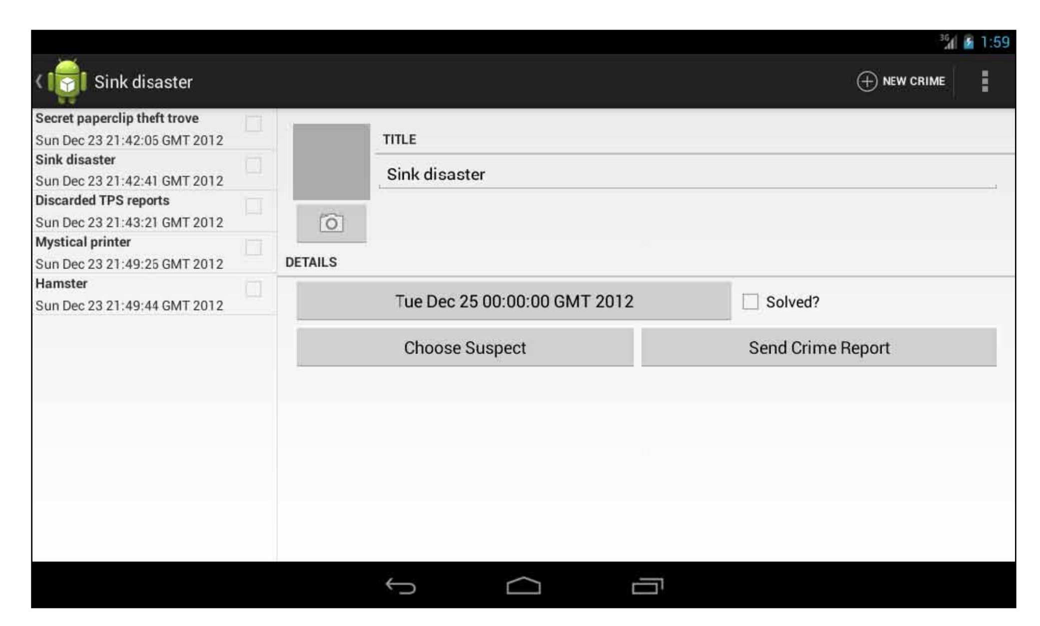


Рис. 22.1. Главное и детализированное представления одновременно находятся на экране

Для тестирования программ этой главы вам понадобится планшетное устройство или AVD. Чтобы создать виртуальный планшет AVD, выполните команду Window ▸ Android Virtual Device Manager. Выполните команду New и выберите в качестве устройства (Device) AVD один из двух вариантов, выделенных на рис. 22.2. Затем задайте для AVD целевой API уровня 17.

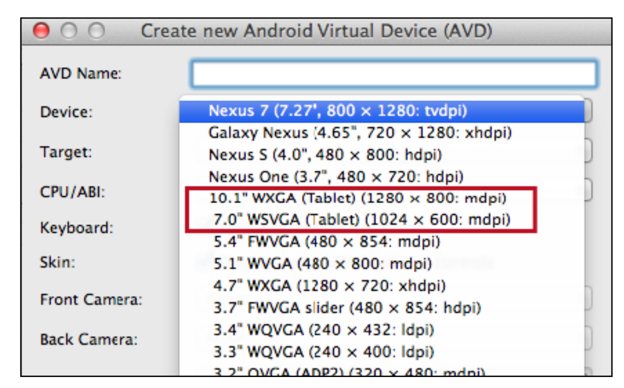


Рис. 22.2. Выбор устройства для планшетного AVD

Гибкость макета

На телефоне активность `CrimeListActivity` должна заполнять однопанельный макет, как она делает в настоящее время. На планшете она должна заполнять двухпанельный макет, способный одновременно отображать главное и детализированное представления.

В двухпанельном макете `CrimeListActivity` будет отображать как `CrimeListFragment`, так и `CrimeFragment`, как показано на рис. 22.3.

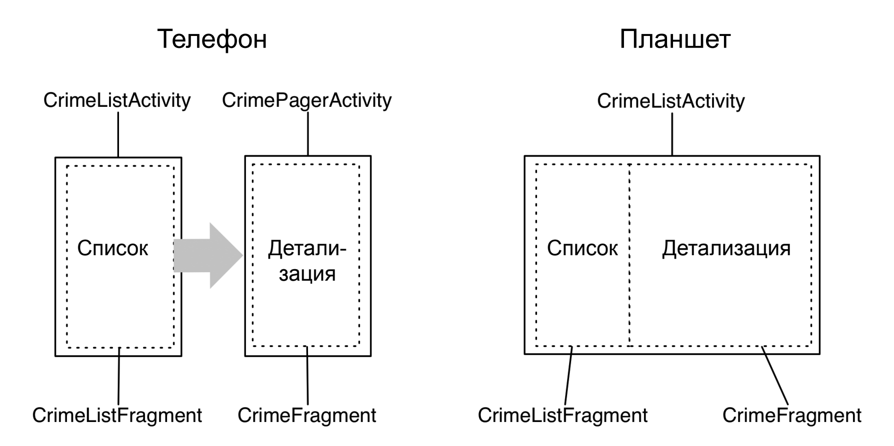


Рис. 22.3. Разновидности макетов

Для этого необходимо:

- изменить `SingleFragmentActivity`, чтобы выбор заполняемого макета не был жестко фиксирован в программе;
- создать новый макет, состоящий из двух контейнеров фрагментов;
- изменить `CrimeListActivity`, чтобы на телефонах заполнялся однопанельный макет, а на планшетах — двухпанельный.

Модификация SingleFragmentActivity

`CrimeListActivity` является субклассом `SingleFragmentActivity`. В настоящее время класс `SingleFragmentActivity` настроен таким образом, чтобы он всегда заполнял activity_fragment.xml.

Чтобы класс `SingleFragmentActivity` стал более гибким, мы сделаем так, чтобы субкласс мог предоставлять свой идентификатор ресурса макета.

В файле SingleFragmentActivity.java добавьте защищенный метод, который возвращает идентификатор макета, заполняемого активностью.

Листинг 22.1. Обеспечение гибкости SingleFragmentActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    protected int getLayoutResId() {
        return R.layout.activity_fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        setContentView(getLayoutResId());
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

Реализация класса `SingleFragmentActivity` по умолчанию будет работать так же, как прежде, но теперь его субклассы могут переопределить `getLayoutResId()` для возвращения макета, отличного от activity_fragment.xml.

Создание макета с двумя контейнерами фрагментов

На панели Package Explorer щелкните правой кнопкой мыши на каталоге res/layout/ и создайте новый файл Android в формате XML. Убедитесь в том, что для файла выбран тип ресурса `Layout`, присвойте файлу имя activity_twopane.xml и назначьте его корневым элементом `LinearLayout`.

Напишите разметку XML для двухпанельного макета, изображенного на рис. 22.4.

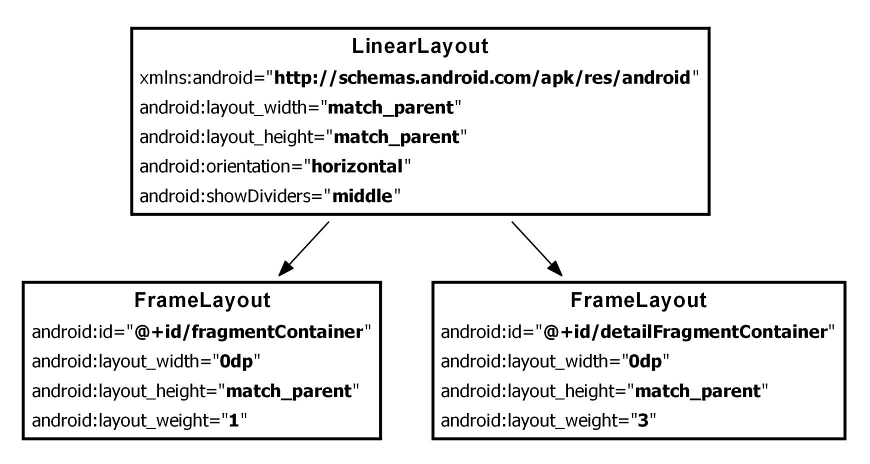


Рис. 22.4. Макет с двумя контейнерами фрагментов (layout/activity_twopane.xml)

Обратите внимание: у первого виджета `FrameLayout` задан идентификатор макета `fragmentContainer`, поэтому код `SingleFragmentActivity.onCreate(…)` может работать так же, как прежде. При создании активности фрагмент, возвращаемый `createFragment()`, появится на левой панели.

Протестируйте макет в `CrimeListActivity`; для этого переопределите метод `getLayoutResId()` так, чтобы он возвращал `R.layout.activity_twopane`.

Листинг 22.2. Переход к файлу двухпанельного макета (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }
}
```

Запустите приложение CriminalIntent на планшетном устройстве и убедитесь в том, что на экране отображаются две панели (рис. 22.5). Большая панель детализации

пуста, а нажатие на элементе списка не отображает подробную информацию о преступлении. Контейнер детализированного представления будет подключен позднее в этой главе.

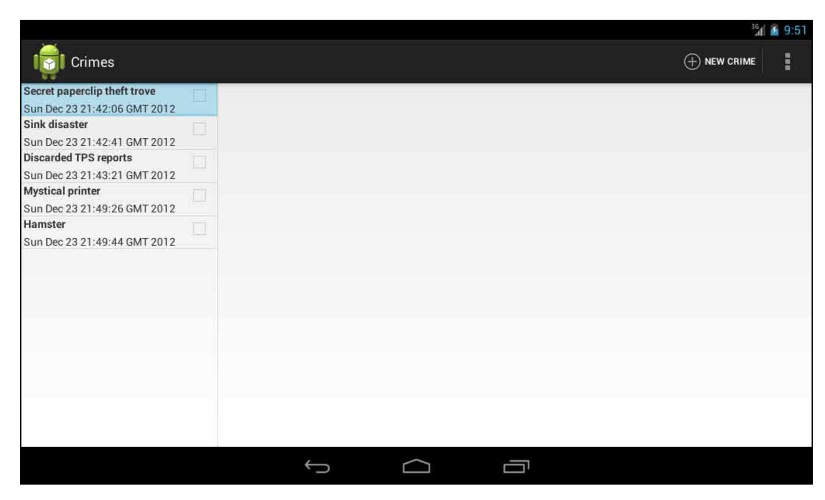


Рис. 22.5. Двухпанельный макет на планшете

В своей текущей версии `CrimeListActivity` также заполняет двухпанельный интерфейс при запуске на телефоне. В следующем разделе мы исправим этот недостаток при помощи ресурса-псевдонима.

Использование ресурса-псевдонима

Ресурс-псевдоним (alias resource) представляет собой ресурс, указывающий на другой ресурс. Ресурсы-псевдонимы находятся в каталоге res/values/ и по умолчанию определяются в файле refs.xml.

В этом разделе мы создадим ресурс-псевдоним, который на телефонах ссылается на макет activity_fragment.xml, и на планшетах — на макет activity_twopane.xml.

На панели Package Explorer щелкните правой кнопкой мыши на каталоге res/layout/ и создайте новый файл Android в формате XML. Убедитесь в том, что для файла выбран тип ресурса `Values`, присвойте файлу имя refs.xml, назначьте его корневым элементом `resources` и щелкните на кнопке Finish. Затем добавьте элемент, приведенный в листинге 22.3.

Листинг 22.3. Создание значения по умолчанию для ресурса-псевдонима (res/values/refs.xml)

```
<resources>

  <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>

</resources>
```

Значение ресурса представляет собой ссылку на однопанельный макет. Ресурс также обладает идентификатором: `R.layout.activity_masterdetail`. Обратите внимание:

внутренний класс идентификатора определяется атрибутом type псевдонима. И хотя сам псевдоним находится в res/values/, его идентификатор хранится в `R.layout`.

Теперь этот идентификатор ресурса может использоваться вместо `R.layout.activity_fragment`. Внесите следующее изменение в `CrimeListActivity`.

Листинг 22.4. Повторная замена макета (CrimeListActivity.java)

```
@Override
protected int getLayoutResId() {
    return R.layout.activity_twopane;
    return R.layout.activity_masterdetail;
}
```

Запустите приложение CriminalIntent и убедитесь в том, что псевдоним работает правильно. Активность `CrimeListActivity` снова должна заполнять однопанельный макет.

Создание альтернативы для планшета

Так как псевдоним находится в res/values/, он используется по умолчанию. Следовательно, по умолчанию `CrimeListActivity` заполняет однопанельный макет.

Теперь мы создадим альтернативный ресурс, чтобы псевдоним `activity_masterdetail` на планшетных устройствах ссылался на activity_twopane.xml.

На панели Package Explorer щелкните правой кнопкой мыши на каталоге res/ и создайте новую папку с именем values-sw600dp. Скопируйте файл res/values/refs.xml в res/values-sw600dp/ и измените макет, на который ссылается псевдоним.

Листинг 22.5. Альтернативный псевдоним для планшетных устройств (res/values-sw600dp/refs.xml)

```
<resources>

  <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
  <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>

</resources>
```

Что означает конфигурационный квалификатор `-sw600dp`? Сокращение «sw» происходит от «Smallest Width» (наименьшая ширина), но относится к меньшему размеру экрана, а следовательно, не зависит от текущей ориентации устройства.

Используя квалификатор `-sw600dp`, вы указываете: «Этот ресурс должен использоваться на любых устройствах, у которых меньший размер составляет 600dp и выше». Это хороший критерий для определения экранов планшетных устройств.

И еще одно: квалификатор sw появился в Android 3.2. Это означает, что планшетное устройство с Android 3.0 и 3.1 его не опознает. Для решения этой проблемы можно добавить еще один альтернативный ресурс, использующий устаревший квалификатор размера экрана `-xlarge`.

Щелкните правой кнопкой мыши на каталоге res/ и создайте новую папку с именем values-xlarge. Скопируйте файл res/values-sw600dp/refs.xml в res/values-xlarge/. Теперь

у вас имеется другой альтернативный ресурс, который выглядит так, как показано в листинге 22.6.

Листинг 22.6. Альтернативный псевдоним для планшетных устройств до версии 3.2 (res/values-xlarge/refs.xml)

```
<resources>

  <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>

</resources>
```

Квалификатор -xlarge содержит ресурсы для устройств с минимальными размерами 720 × 960dp. Он будет использоваться только устройствами с версией, предшествующей Android 3.2. Более поздние версии обнаружат и используют -sw600dp.

Запустите приложение CriminalIntent на телефоне и планшете. Убедитесь в том, что одно- и двухпанельные макеты отображаются там, где предполагалось.

Активность: управление фрагментами

Итак, макеты ведут себя так, как положено, и мы можем перейти к добавлению CrimeFragment в контейнер фрагмента детализации при использовании двухпанельного макета CrimeListActivity.

На первый взгляд кажется, что для этого достаточно написать альтернативную реализацию CrimeListFragment.onListItemClick(…) для планшетов. Вместо запуска нового экземпляра CrimePagerActivity метод onListItemClick(…) получает экземпляр FragmentManager, принадлежащий CrimeListActivity, и закрепляет транзакцию, которая добавляет CrimeFragment в контейнер фрагмента детализации.

Код выглядит примерно так:

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Получение экземпляра Crime от адаптера
    Crime crime = ((CrimeAdapter)getListAdapter()).getItem(position);
    // Включение нового экземпляра CrimeFragment в макете активности
    Fragment fragment = CrimeFragment.newInstance(crime.getId());
    FragmentManager fm = getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detailFragmentContainer, fragment)
        .commit();
}
```

Такое решение работает, но оно противоречит хорошему стилю программирования Android. Предполагается, что фрагменты представляют собой автономные компонуемые блоки. Если написанный вами фрагмент добавляет фрагменты в FragmentManager активности, значит, он делает допущения относительно того, как работает активность-хост, и перестает быть автономным компонуемым блоком.

Например, в приведенном выше коде CrimeListFragment добавляет CrimeFragment в CrimeListActivity и предполагает, что в макете CrimeListActivity присутствует контейнер detailFragmentContainer. Такими вопросами должна заниматься активность-хост CrimeListFragment, а не CrimeListFragment.

Для сохранения независимости фрагментов мы делегируем выполнение работы активности-хосту, определяя интерфейсы обратного вызова в ваших фрагментах. Активности-хосты реализуют эти интерфейсы для выполнения операций по управлению фрагментами и обеспечения макетно-зависимого поведения.

Интерфейсы обратного вызова фрагментов

Чтобы делегировать функциональность активности-хосту, фрагмент обычно определяет интерфейс обратного вызова с именем `Callbacks`. Этот интерфейс определяет работу, которая должна быть выполнена для фрагмента его «начальником» — активностью-хостом. Любая активность, выполняющая функции хоста фрагментов, должна реализовать этот интерфейс.

С интерфейсом обратного вызова фрагмент может вызывать методы активности-хоста, не располагая никакой информацией о ней.

Реализация CrimeListFragment.Callbacks

Чтобы реализовать интерфейс `Callbacks`, следует сначала определить переменную для хранения объекта, реализующего `Callbacks`. Затем активность-хост преобразуется к `Callbacks`, а результат присваивается этой переменной.

Активность назначается в методе жизненного цикла `Fragment`:

```
public void onAttach(Activity activity)
```

Этот метод вызывается при присоединении фрагмента к активности (независимо от того, было он сохранен или нет). Аналогичным образом переменной присваивается `null` в соответствующем завершающем методе жизненного цикла:

```
public void onDetach()
```

Переменной присваивается `null`, потому что в дальнейшем вы не сможете обратиться к активности или рассчитывать на то, что активность продолжит существовать.

В файле CrimeListFragment.java включите в класс `CrimeListFragment` интерфейс `Callbacks`. Также добавьте переменную `mCallbacks` и переопределите методы `onAttach(Activity)` и `onDetach()`, в которых задается и сбрасывается ее значение.

Листинг 22.7. Добавление интерфейса обратного вызова (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;
    private boolean mSubtitleVisible;
    private Callbacks mCallbacks;

    /**
     * Обязательный интерфейс для активности-хоста.
     */
    public interface Callbacks {
        void onCrimeSelected(Crime crime);
    }

    @Override
```

```
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        mCallbacks = (Callbacks)activity;
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }
```

Теперь у CrimeListFragment имеется механизм вызова методов активности-хоста. Неважно, какая активность является хостом — если она реализует CrimeListFragment.Callbacks, внутренняя реализация CrimeListFragment будет работать одинаково.

Обратите внимание на то, как CrimeListFragment выполняет непроверяемое преобразование своей активности к CrimeListFragment.Callbacks. Это означает, что активность-хост *должна* реализовать CrimeListFragment.Callbacks. В самой зависимости нет ничего плохого, но ее важно документировать.

Затем в классе CrimeListActivity реализуйте CrimeListFragment.Callbacks. Метод onCrimeSelected(Crime) пока оставьте пустым.

Листинг 22.8. Реализация обратных вызовов (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }

    public void onCrimeSelected(Crime crime) {
    }
}
```

CrimeListFragment будет вызывать этот метод onListItemClick(…), а также тогда, когда пользователь выбирает команду создания новой записи преступления. Для начала нужно понять, как должна быть устроена реализация CrimeListActivity.onCrimeSelected(Crime).

При вызове onCrimeSelected(Crime) класс CrimeListActivity должен выполнить одну из двух операций:

- если используется телефонный интерфейс — запустить новый экземпляр CrimePagerActivity;
- если используется планшетный интерфейс — поместить CrimeFragment в detailFragmentContainer.

Чтобы определить, интерфейс какого типа был заполнен, можно проверить конкретный идентификатор интерфейса. Впрочем, лучше проверить наличие в макете detailFragmentContainer. Такая проверка будет более точной и надежной. Имена файлов могут изменяться, и вас на самом деле не интересует, по какому файлу заполнялся макет; необходимо знать лишь то, имеется ли у него контейнер detailFragmentContainer для размещения CrimeFragment.

Если макет содержит detailFragmentContainer, мы создадим транзакцию фрагмента, которая удаляет существующий экземпляр CrimeFragment из detailFragmentContainer (если он имеется) и добавляет экземпляр CrimeFragment, который мы хотим там видеть.

В файле CrimeListActivity.java реализуйте метод onCrimeSelected(Crime), который будет обрабатывать выбор преступления в любом варианте интерфейса.

Листинг 22.9. Условный запуск CrimeFragment (CrimeListActivity.java)

```
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detailFragmentContainer) == null) {
        // Запуск экземпляра CrimePagerActivity
        Intent i = new Intent(this, CrimePagerActivity.class);
        i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
        startActivity(i);
    } else {
        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        Fragment oldDetail = fm.findFragmentById(R.id.detailFragmentContainer);
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());
        if (oldDetail != null) {
            ft.remove(oldDetail);
        }
        ft.add(R.id.detailFragmentContainer, newDetail);
        ft.commit();
    }
}
```

Наконец, в классе CrimeListFragment мы будем вызывать onCrimeSelected(Crime) в тех местах, где сейчас запускается новый экземпляр CrimePagerActivity.

В файле CrimeListFragment.java измените методы onListItemClick(…) и onOptionsItemSelected(MenuItem) так, чтобы в них вызывался метод Callbacks.onCrimeSelected(Crime).

Листинг 22.10. Активизация обратных вызовов (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Получение объекта Crime от адаптера
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    // Запуск экземпляра CrimePagerActivity
    Intent i = new Intent(getActivity(), CrimePagerActivity.class);
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
    startActivity(i);
    mCallbacks.onCrimeSelected(c);
}
```

```
...

@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
            startActivity(i);
            ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
            mCallbacks.onCrimeSelected(crime);
            return true;
        ...
    }
}
```

При обратном вызове в `onOptionsItemSelected(…)` содержимое списка также немедленно перезагружается после добавления нового преступления. Это необходимо, потому что на планшетах при добавлении нового преступления список остается видимым на экране (прежде его закрывал экран детализации).

Запустите приложение CriminalIntent на планшете. Создайте новое преступление; экземпляр `CrimeFragment` добавляется и отображается в `detailFragmentContainer`. Затем просмотрите какое-либо старое преступление и убедитесь в том, что `CrimeFragment` заменяется новым экземпляром.

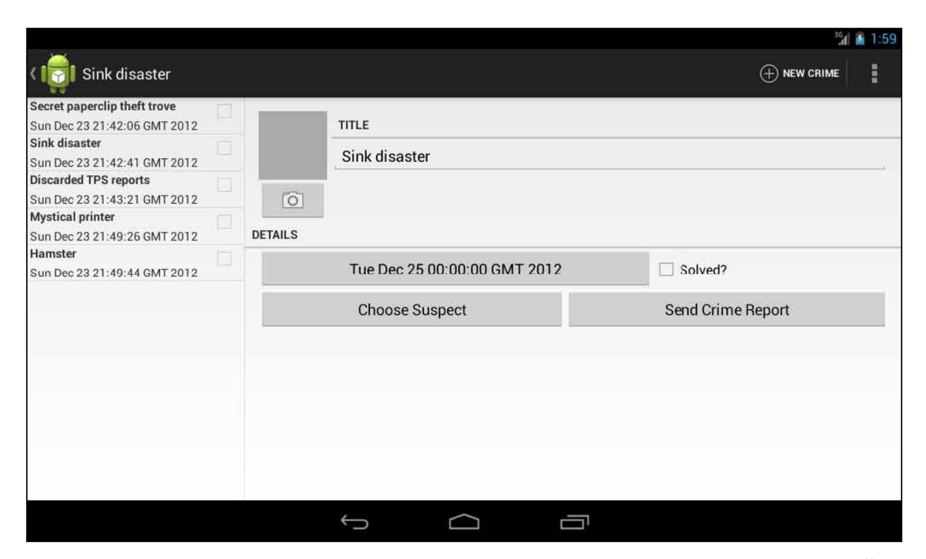


Рис. 22.6. Главное и детализированное представления связаны между собой

Однако внесение изменений в преступление не приводит к обновлению списка. В настоящее время список перезагружается только после добавления преступления и в `CrimeListFragment.onResume()`. При этом на планшетах экземпляр

CrimeListFragment остается видимым рядом с CrimeFragment. CrimeListFragment не приостанавливается при появлении CrimeFragment, поэтому и возобновления не происходит.

Для решения этой проблемы мы используем другой интерфейс обратного вызова из CrimeFragment.

Реализация CrimeFragment.Callbacks

CrimeFragment определяет следующий интерфейс:

```
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
```

CrimeFragment будет вызывать метод onCrimeUpdated(Crime) активности-хоста при сохранении любых изменений в Crime. Реализация onCrimeUpdated(Crime) в CrimeListActivity перезагружает список CrimeListFragment.

Прежде чем браться за интерфейс в CrimeFragment, добавьте в CrimeListFragment метод, который вызывается для перезагрузки списка CrimeListFragment.

Листинг 22.11. Добавление метода updateUI() (CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {
...
public void updateUI() {
    ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
}
```

В файле CrimeFragment.java добавьте интерфейс обратного вызова, а также поле mCallbacks и реализации onAttach(…) и onDetach().

Листинг 22.12. Добавление обратных вызовов CrimeFragment (CrimeFragment.java)

```
...
private ImageView mPhotoView;
private Button mSuspectButton;
private Callbacks mCallbacks;

/**
 * Обязательный интерфейс для активности-хоста
 */
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallbacks = (Callbacks)activity;
}

@Override
public void onDetach() {
    super.onDetach();
```

```
    mCallbacks = null;
}

public static CrimeFragment newInstance(UUID crimeId) {
    ...
}
```

Затем реализуйте `CrimeFragment.Callbacks` в `CrimeListActivity`, чтобы список перезагружался в `onCrimeUpdated(Crime)`.

Листинг 22.13. Обновление списка преступлений (CrimeListActivity.java)

```
public void onCrimeUpdated(Crime crime) {
    FragmentManager fm = getSupportFragmentManager();
    CrimeListFragment listFragment = (CrimeListFragment)
            fm.findFragmentById(R.id.fragmentContainer);
    listFragment.updateUI();
}
```

В файле CrimeFragment.java добавьте вызовы `onCrimeUpdated(Crime)` при изменении заголовка или флага раскрытия `Crime`.

Листинг 22.14. Вызов onCrimeUpdated(Crime) (CrimeFragment.java)

```
@Override
@TargetApi(11)
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    ...

    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        public void onTextChanged(CharSequence c, int start, int before, int count)
{
            mCrime.setTitle(c.toString());
            mCallbacks.onCrimeUpdated(mCrime);
            getActivity().setTitle(mCrime.getTitle());
        }
        ...
    });

    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
            // Задание признака раскрытия преступления
            mCrime.setSolved(isChecked);
            mCallbacks.onCrimeUpdated(mCrime);
        }
    });
    ...

    return v;
}
```

Также необходимо вызвать onCrimeUpdated(Crime) в методе onActivityResult(…), где могут измениться дата, фотография и подозреваемый в преступлении. В настоящее время фотография и подозреваемый не отображаются в представлении элемента списка, но класс CrimeFragment все равно должен вести себя корректно и сообщать об этих обновлениях.

Листинг 22.15. Вызов onCrimeUpdated(Crime) (№2) (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        Date date = (Date)data.getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mCallbacks.onCrimeUpdated(mCrime);
        updateDate();
    } else if (requestCode == REQUEST_PHOTO) {
        // Создание нового объекта Photo и связывание его с Crime
        String filename = data
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename != null) {
            Photo p = new Photo(filename);
            mCrime.setPhoto(p);
            mCallbacks.onCrimeUpdated(mCrime);
            showPhoto();
        }
    } else if (requestCode == REQUEST_CONTACT) {
        ...

        c.moveToFirst();
        String suspect = c.getString(0);
        mCrime.setSuspect(suspect);
        mCallbacks.onCrimeUpdated(mCrime);
        mSuspectButton.setText(suspect);
        c.close();
    }
}
```

CrimeListActivity теперь содержит реализацию CrimeFragment.Callbacks. Однако при попытке запустить CriminalIntent на телефоне произойдет сбой. Помните: любая активность, являющаяся хостом CrimeFragment, должна реализовать CrimeFragment.Callbacks. Необходимо реализовать CrimeFragment.Callbacks в CrimePagerActivity.

Для CrimePagerActivity достаточно пустой реализации, в которой onCrimeUpdated(Crime) не делает ничего. Когда CrimePagerActivity является хостом CrimeFragment, необходимая перезагрузка списка уже выполняется в onResume().

Листинг 22.16. Пустая реализация CrimeFragment.Callbacks (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity
    implements CrimeFragment.Callbacks {
    ...

    public void onCrimeUpdated(Crime crime) {
    }
}
```

Запустите приложение CriminalIntent на планшете и убедитесь в том, что `ListView` обновляется при внесении изменений в `CrimeFragment`. Затем запустите его на телефоне и убедитесь в том, что приложение продолжает работать, как прежде.

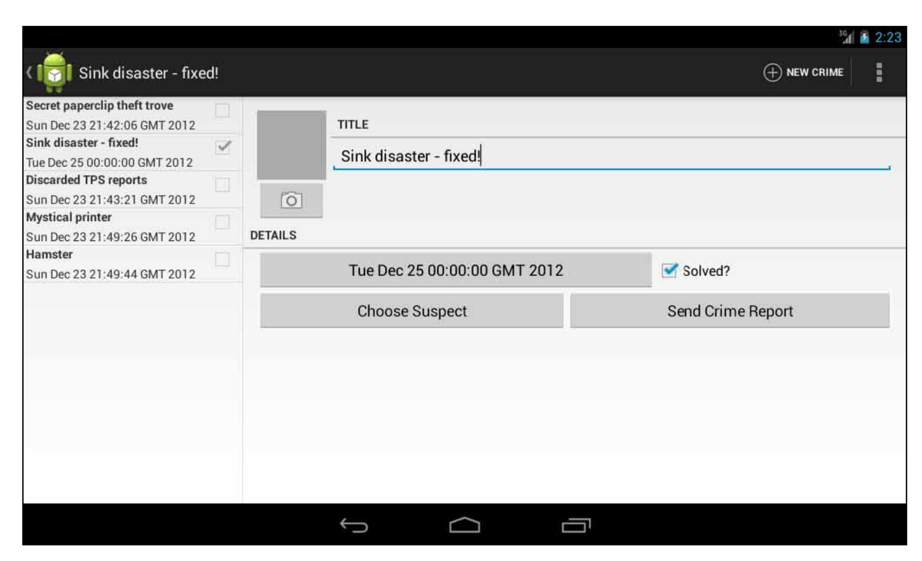


Рис. 22.7. Изменения, внесенные в детализированном представлении, отражаются в списке

На этом наше знакомство с CriminalIntent подходит к концу. На протяжении 13 глав мы создали сложное приложение, которое использует фрагменты, взаимодействует с другими приложениями, делает снимки и сохраняет данные. Почему бы не отпраздновать окончание работы куском торта? Только не забудьте убрать за собой крошки, чтобы ваш проступок не попал в CriminalIntent.

Для любознательных: подробнее об определении размера экрана

До выхода Android 3.2 для предоставления альтернативных ресурсов в зависимости от размера устройства использовался квалификатор размера экрана. Этот квалификатор группирует разные устройства на четыре категории — `small`, `normal`, `large` и `xlarge`. В табл. 22.1 приведены минимальные размеры для каждого квалификатора.

Таблица 22.1. Квалификаторы размера экрана

Имя	Минимальный размер экрана
small	320 x 426dp
normal	320 x 470dp
large	480 x 640dp
xlarge	720 x 960dp

Квалификаторы размера экрана были объявлены устаревшими в Android 3.2. Они были заменены квалификаторами, позволяющими определять размеры устройства. В табл. 22.2 перечислены эти новые квалификаторы.

Таблица 22.2. Дискретные квалификаторы размера экрана

Формат квалификатора	Описание
wXXXdp	Доступная ширина: ширина больше либо равна XXX dp
hXXXdp	Доступная высота: высота больше либо равна XXX dp
swXXXdp	Минимальная ширина: ширина или высота (меньшая из двух) больше либо равна XXX dp

Предположим, вы хотите задать макет, который должен использоваться только в том случае, если ширина экрана не менее 300dp. В этом случае можно поместить файл макета в каталог res/layout-w300dp («w» — сокращение от «width», то есть «ширина»). То же самое можно сделать для высоты при помощи префикса «h» («Height», то есть «высота»).

Впрочем, ширина и высота могут меняться местами в зависимости от ориентации устройства. Для обнаружения конкретного размера экрана используется префикс «sw» («Smallest Width», то есть «минимальная ширина»). Он задает наименьший размер экрана, которым в зависимости от ориентации устройства может быть как ширина, так и высота. Если размеры экрана равны 1024 × 800, то метрика `sw` равна 800. Если размеры экрана равны 800 × 1024, то метрика `sw` все равно равна 800.

23 Подробнее об интентах и задачах

В этой главе мы используем неявные интенты для создания приложения-лаунчера, заменяющего стандартный лаунчер Android. Чтобы приложение работало правильно, нам придется углубить свое понимание интентов, фильтров интентов и схем взаимодействий между приложениями в среде Android.

Создание приложения NerdLauncher

Создайте новый проект (New ▸ Android Application Project) с теми же параметрами, которые использовались для CriminalIntent (рис. 23.1). Присвойте проекту имя NerdLauncher и создайте его в пакете `com.bignerdranch.android.nerdlauncher`.

Создайте активность, но без пользовательского значка лаунчера. Выберите создание новой пустой активности. Присвойте активности имя `NerdLauncherActivity` и щелкните на кнопке Finish.

Класс `NerdLauncherActivity` должен быть субклассом `SingleFragmentActivity`, поэтому этот класс необходимо добавить в проект. На панели Package Explorer найдите файл SingleFragmentActivity.java в пакете `CriminalIntent`. Скопируйте его в пакет `com.bignerdranch.android.nerdlauncher`. При копировании файлов Eclipse автоматически обновляет объявления пакетов.

Нам также понадобится макет activity_fragment.xml. Скопируйте res/layout/activity_fragment.xml в каталог res/layout проекта NerdLauncher.

NerdLauncher будет отображать список приложений на устройстве. Пользователь нажимает элемент списка, чтобы запустить соответствующее приложение. А теперь посмотрим, какие объекты для этого понадобятся.

Рис. 23.1. Создание проекта NerdLauncher

Класс `NerdLauncherFragment` является субклассом `ListFragment`, а в качестве представления он будет использовать стандартный класс `ListView`, прилагаемый к `ListFragment`.

Создайте новый класс с именем `NerdLauncherFragment` и назначьте его суперклассом `android.support.v4.app.ListFragment`. Пока оставьте этот класс пустым.

Откройте файл NerdLauncherActivity.java и измените суперкласс `NerdLauncherActivity` на `SingleFragmentActivity`. Удалите код шаблона и переопределите метод `createFragment()` так, чтобы он возвращал `NerdLauncherFragment`.

Листинг 23.1. Субкласс SingleFragmentActivity (NerdLauncherActivity.java)

```
public class NerdLauncherActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nerd_launcher);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_nerd_launcher, menu);
        return true;
    }

    @Override
    public Fragment createFragment() {
        return new NerdLauncherFragment();
    }
}
```

Обработка неявного интента

NerdLauncher отображает список приложений на устройстве. Для этого NerdLauncher отправляет неявный интент, на который должна отреагировать главная активность каждого приложения. В интент включается действие MAIN и категория LAUNCHER. Вы уже видели следующий фильтр интентов в своих проектах:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

В файле NerdLauncherFragment.java переопределите метод onCreate(Bundle) для создания неявного интента. Получите от PackageManager список активностей, соответствующих интенту. Пока мы ограничимся простой регистрацией количества активностей, возвращенных PackageManager.

Листинг 23.2. Получение информации у PackageManager (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends ListFragment {
    private static final String TAG = "NerdLauncherFragment";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

        Log.i(TAG, "I've found " + activities.size() + " activities.");
    }
}
```

Запустите приложение NerdLauncher и посмотрите в данных LogCat, сколько приложений вернул экземпляр PackageManager.

В CriminalIntent для отправки отчетов использовался неявный интент. Чтобы представить на экране список выбора приложений, мы создали неявный интент, упаковали его в объект выбора и отправили ОС вызовом startActivity(Intent):

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Создание и размещение дополнений интентов
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

Почему мы не используем этот подход здесь? Вкратце дело в том, что фильтр интентов MAIN/LAUNCHER может соответствовать или не соответствовать неявному интенту MAIN/LAUNCHER, отправленному из startActivity(…).

Оказывается, вызов startActivity(Intent) не означает «Запустить активность, соответствующую этому неявному интенту». Он означает «Запустить активность по умолчанию соответствующую этому неявному интенту». Когда вы отправляете

неявный интент с использованием startActivity(…) (или startActivityForResult(…)), ОС незаметно включает в интент категорию Intent.CATEGORY_DEFAULT.

Таким образом, если вы хотите, чтобы фильтр интентов соответствовал неявным интентам, отправленным через startActivity(…), вы должны включить в этот фильтр интентов категорию DEFAULT.

Активность с фильтром интентов MAIN/LAUNCHER является главной точкой входа приложения, которому она принадлежит. Для нее важно лишь то, что она является главной точкой входа приложения, а является ли она главной точкой входа «по умолчанию» — несущественно, поэтому она не обязана включать категорию CATEGORY_DEFAULT.

Так как фильтры интентов MAIN/LAUNCHER могут не включать CATEGORY_DEFAULT, надежность их соответствия неявным интентам, отправленным вызовом startActivity(…), не гарантирована. Поэтому мы используем интент для прямого запроса у PackageManager информации об активностях с фильтром интентов MAIN/LAUNCHER.

Следующий шаг — отображение меток этих активностей в списке ListView экземпляра NerdLauncherFragment. *Метка* (label) активности представляет собой отображаемое имя — нечто, понятное пользователю. Если учесть, что эти активности являются активностями лаунчера, такой меткой, скорее всего, должно быть имя приложения.

Метки активностей вместе с другими метаданными содержатся в объектах ResolveInfo, возвращаемых PackageManager.

Сначала добавьте следующий код для сортировки объектов ResolveInfo, возвращаемых PackageManager, в алфавитном порядке меток, получаемых методом ResolveInfo.loadLabel(…).

Листинг 23.3. Алфавитная сортировка (NerdLauncherFragment.java)

```
...
Log.i("NerdLauncher", "I've found " + activities.size() + " activities.");

Collections.sort(activities, new Comparator<ResolveInfo>() {
    public int compare(ResolveInfo a, ResolveInfo b) {
        PackageManager pm = getActivity().getPackageManager();
        return String.CASE_INSENSITIVE_ORDER.compare(
            a.loadLabel(pm).toString(),
            b.loadLabel(pm).toString());
    }
});
```

Затем создайте объект ArrayAdapter, который создаст простые представления элементов списка с метками активностей, и назначьте этот адаптер ListView.

Листинг 23.4. Создание адаптера (NerdLauncherFragment.java)

```
...
Collections.sort(activities, new Comparator<ResolveInfo></ResolveInfo>() {
    ...
    }
});
```

```
ArrayAdapter<ResolveInfo> adapter = new ArrayAdapter<ResolveInfo>(
        getActivity(), android.R.layout.simple_list_item_1, activities) {
    public View getView(int pos, View convertView, ViewGroup parent) {
        View v = super.getView(pos, convertView, parent);
        // В документации сказано, что simple_list_item_1
        // является TextView; преобразуем для задания текстового значения.
        TextView tv = (TextView)v;
        ResolveInfo ri = getItem(pos);
        tv.setText(ri.loadLabel(pm));
        return v;
    }
};

setListAdapter(adapter);
```

Запустите приложение NerdLauncher; вы увидите список `ListView`, заполненный метками активностей.

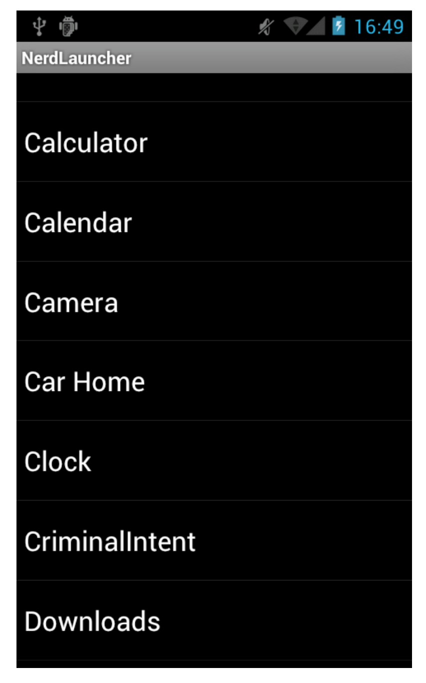


Рис. 23.2. Список активностей

Создание явных интентов на стадии выполнения

Мы использовали неявный интент для сбора информации об активностях и выводе ее в формате списка. Следующим шагом должен стать запуск выбранной активности при нажатии пользователем на элементе списка. Для запуска активности будет использоваться явный интент.

Для создания явного интента нам потребуются дополнительные данные из ResolveInfo — в частности, имя пакета и имя класса активности. Эти данные можно получить из части ResolveInfo с именем ActivityInfo. (О том, какие данные доступны в разных частях ResolveInfo, можно узнать из документации.)

В файле NerdLauncherFragment.java переопределите метод onListItemClick(…) для получения объекта ActivityInfo для элемента списка. Затем используйте его данные для создания явного интента, запускающего активность.

Листинг 23.5. Реализация onListItemClick(…) (NerdLauncherFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    ResolveInfo resolveInfo = (ResolveInfo)l.getAdapter().getItem(position);
    ActivityInfo activityInfo = resolveInfo.activityInfo;

    if (activityInfo == null) return;

    Intent i = new Intent(Intent.ACTION_MAIN);
    i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);

    startActivity(i);
}
```

Обратите внимание: в этом интенте мы отправляем действие как часть явного интента. Большинство приложений ведет себя одинаково независимо от того, включено действие или нет, однако некоторые приложения могут изменять свое поведение. Одна и та же активность может отображать разные интерфейсы в зависимости от того, как она была запущена. Вам как программисту лучше всего четко объявить свои намерения и позволить активностям запуститься так, как они считают нужным.

В листинге 23.5 мы получаем имя пакета и имя класса из метаданных и используем их для создания явной активности методом Intent:

```
public Intent setClassName(String packageName, String className)
```

Этот способ отличается от того, который использовался нами для создания явных интентов в прошлом. Ранее мы использовали конструктор Intent, получающий объекты Context и Class:

```
public Intent(Context packageContext, Class<?> cls)
```

Этот конструктор использует свои параметры для получения ComponentName — имени пакета, объединенного с именем класса. Когда вы передаете Activity и Class для создания Intent, конструктор определяет полное имя пакета по Activity.

Также можно самостоятельно создать ComponentName по именам пакета и класса и использовать следующий метод Intent для создания явного интента:

```
public Intent setComponent(ComponentName component)
```

Однако решение с методом setClassName(…), автоматически создающим имя компонента, получается более компактным.

Запустите NerdLauncher и посмотрите, как работает запуск приложений.

Задачи и стек возврата

Android использует задачи для отслеживания текущего состояния пользователя в каждом выполняемом приложении. *Задача* (task) представляет собой стек активностей, с которыми имеет дело пользователь. Активность в нижней позиции стека называется *базовой активностью*, а активность в верхней позиции видна пользователю. При нажатии кнопки Back верхняя активность извлекается из стека. Если нажать кнопку Back при просмотре базовой активности, вы вернетесь к домашнему экрану.

Диспетчер задач позволяет переключаться между задачами без изменения состояния каждой задачи. Например, если вы начинаете вводить новый контакт и переключаетесь на проверку своей публикации в Твиттере, будут запущены две задачи. При переключении на редактирование контактов сохраняется ваша текущая позиция в обеих задачах.

Иногда запускаемая активность должна добавляться к текущей задаче. В других случаях она должна запускаться в новой задаче, независимой от запустившей ее активности.

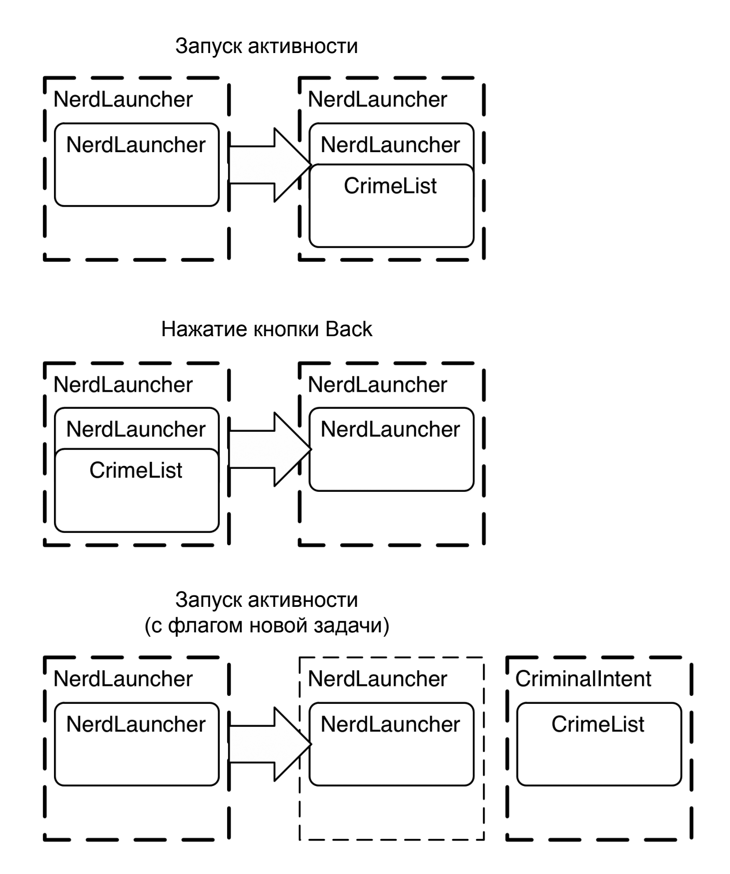


Рис. 23.3. Задачи и стек возврата

По умолчанию новые активности запускаются в текущей задаче. В приложении CriminalIntent все запускавшиеся активности добавлялись к текущей задаче. Это относилось даже к активностям, которые не являлись частью приложения CriminalIntent (например, при запуске активности для отправки отчета). Преимущество добавления активности к текущей задаче заключается в том, что пользователь может выполнять обратную навигацию внутри задачи, а не в иерархии приложений.

В текущей версии все активности, запускаемые из NerdLauncher, добавляются в задачу NerdLaucher. Чтобы убедиться в этом, запустите CriminalIntent из NerdLauncher и вызовите диспетчер задач. (Нажмите кнопку Recents, если она имеется на устройстве; в противном случае используйте долгое нажатие кнопки Home.) Вы не найдете в списке CriminalIntent. Запущенная активность `CrimeListActivity` была добавлена в задачу NerdLauncher. Нажатие на задаче NerdLauncher вернет вас к экрану CriminalIntent, который вы просматривали перед запуском диспетчера задач.

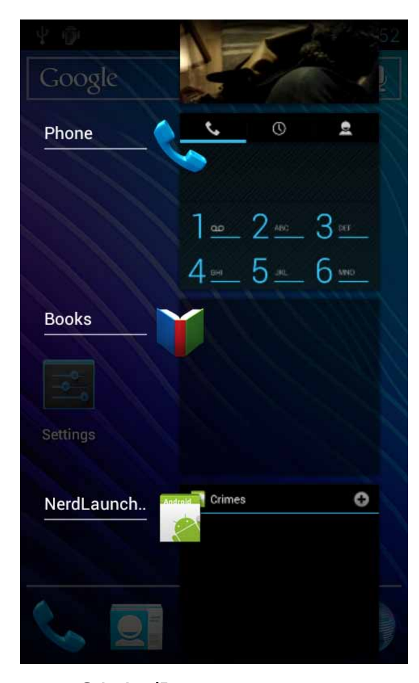


Рис. 23.4. Приложение CriminalIntent не выполняется в отдельной задаче

Мы хотим, чтобы приложение NerdLauncher запускало активности в новых задачах. Далее пользователь может переключаться между выполняемыми приложениями так, как считает нужным. Чтобы при запуске новой активности запускалась новая задача, следует добавить в интент соответствующий флаг.

Запустите приложение NerdLauncher и выберите CriminalIntent. На этот раз при вызове диспетчера задач становится видно, что CriminalIntent выполняется в отдельной задаче.

Листинг 23.6. Добавление флага в интент (NerdLauncherFragment.java)

```
Intent i = new Intent(Intent.ACTION_MAIN);
i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(i);
```

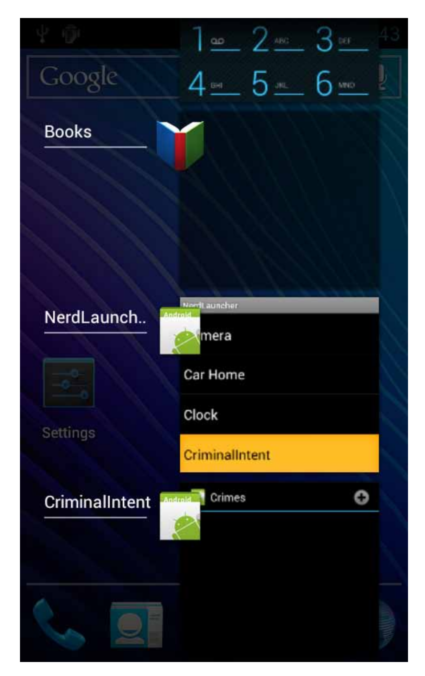


Рис. 23.5. CriminalIntent выполняется в собственной задаче

Повторный запуск CriminalIntent из NerdLauncher не приведет к созданию второй задачи CriminalIntent. Флаг `FLAG_ACTIVITY_NEW_TASK` создает только одну задачу на активность. У `CrimeListActivity` уже имеется работающая задача, поэтому Android переключится на эту задачу вместо запуска новой.

Использование NerdLauncher в качестве домашнего экрана

Но кому захочется запускать приложение, чтобы запускать другие приложения? Гораздо логичнее использовать NerdLauncher как замену для домашнего экрана устройства. Откройте файл AndroidManifest.xml в `NerdLauncher` и добавьте следующий фрагмент в главный фильтр интентов.

Листинг 23.7. Изменение категорий NerdLauncher (AndroidManifest.xml)

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
```

продолжение ⇗

```
  <category android:name="android.intent.category.LAUNCHER" />
  <category android:name="android.intent.category.HOME" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Добавление категорий `HOME` и `DEFAULT` означает, что активность NerdLauncher должна включаться в число вариантов домашнего экрана. Нажмите кнопку Home и вам будет предложено использовать NerdLauncher.

(Если вы назначите NerdLauncher домашним экраном, а потом захотите отменить свой выбор, выполните команду Settings ▸ Applications ▸ Manage Applications. Выберите All, найдите NerdLauncher и сбросьте режим запуска по умолчанию Launch by default. При следующем нажатии кнопки Home вы сможете выбрать новый домашний экран по умолчанию.)

Упражнение. Значки, изменение порядка задач

В этой главе мы использовали метод `ResolveInfo.loadLabel(…)` для отображения содержательных имен в лаунчере. Класс `ResolveInfo` предоставляет аналогичный метод `loadIcon()` для получения значка, отображаемого для каждого приложения. Вам предлагается несложное упражнение: снабдить каждое приложение в NerdLauncher значком.

Если вам захочется большего, добавьте в NerdLauncher другую активность для переключения между выполняемыми задачами. Для этого используйте системную службу `ActivityManager`, которая предоставляет информацию об активностях, задачах и приложениях, выполняемых в настоящее время. В отличие от `PackageManager` класс `Activity` не предоставляет вспомогательного метода `getActivityManager()` для получения доступа к этой системной службе.

Вместо этого для получения объекта `ActivityManager` следует вызвать `Activity.getSystemService()` с передачей в параметре константы `Activity.ACTIVITY_SERVICE`. Вызовите `getRunningTasks()` для получения списка выполняемых задач, упорядоченных по времени запуска (от новых к старым). Чтобы вывести одну из таких задач на передний план, вызовите метод `moveTaskToFront()`. Обязательно сверьтесь со справочной документацией Android — для переключения между задачами в манифест приложения необходимо добавить дополнительное разрешение.

Для любознательных: процессы и задачи

Для существования любого объекта необходима память и виртуальная машина. *Процесс* представляет собой место, созданное ОС, в котором существуют объекты вашего приложения и в котором выполняется само приложение.

Процессам могут принадлежать ресурсы, находящиеся под управлением ОС — память, сетевые сокеты, открытые файлы и т. д. Процесс также содержит минимум один (а вероятно, несколько) программный *поток* (thread). На платформе Android процесс всегда выполняется ровно на одной *виртуальной машине Dalvik*.

Как правило, каждый компонент приложения в Android связывается ровно с одним процессом (хотя встречаются довольно смутные исключения). Приложение создается с собственным процессом, который становится процессом по умолчанию для всех компонентов приложения.

Отдельные компоненты можно назначать разным процессам, но мы рекомендуем придерживаться процесса по умолчанию. Если вы думаете, что какой-то код должен выполняться в другом процессе, аналогичного результата обычно удается добиться с использованием *многопоточности* (multi-threading), которая программируется в Android намного проще, чем многопроцессное выполнение.

Каждый экземпляр активности существует ровно в одном процессе и ровно в одной задаче. Впрочем, на этом все сходство и завершается. Задачи содержат только активности и часто состоят из активностей разных приложений. С другой стороны, процессы содержат только выполняемый код и объекты приложения.

Процессы и задачи легко спутать, потому что эти концепции отчасти перекрываются, а для ссылок на них часто используются имена приложений. Например, при запуске CriminalIntent из NerdLauncher ОС создает процесс `CriminalIntent` и новую задачу, для которой `CrimeListActivity` является базовой активностью. В диспетчере задач эта задача снабжается меткой CriminalIntent.

Задача, в которой существует активность, может быть не связана с процессом, в котором она существует. Когда мы запустили контактное приложение для выбора подозреваемого в CriminalIntent, оно было запущено в задаче CriminalIntent — но при этом выполнялось в процессе контактного приложения.

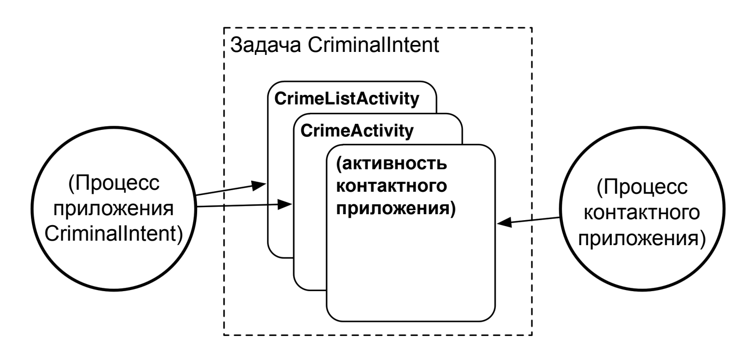


Рис. 23.6. Задачи и процессы

Это означает, что когда пользователь нажимает кнопку Back для перехода между разными активностями, он может незаметно для себя переключаться между процессами.

В этой главе мы создавали задачи и переключались между ними. Как насчет уничтожения задач или замены стандартного диспетчера задач Android? К сожалению, Android не предоставляет средств для решения любой из этих задач. Долгое нажатие на кнопке Home жестко связано с диспетчером задач по умолчанию, а задачи уничтожаться не могут. Процессы, напротив, *могут* уничтожаться. Приложения, рекламируемые в магазине Google Play как уничтожители задач, в действительности являются уничтожителями процессов.

24 Стили и включения

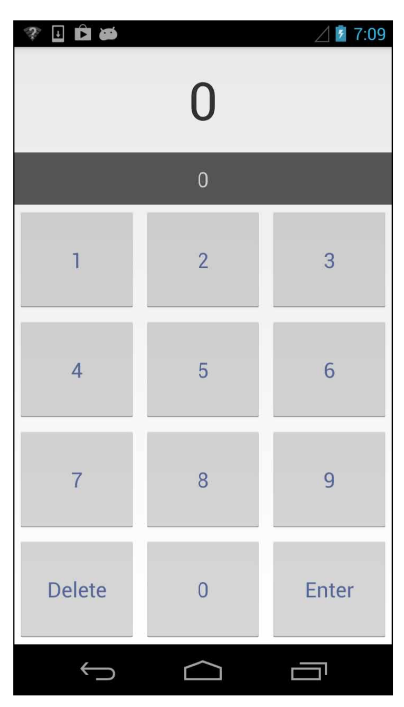


Рис. 24.1. Приложение RemoteControl

Хотя вашей первоочередной целью является нормальное функционирование приложения, не стоит забывать и о том, как приложение смотрится и насколько удобно с ним работать. Рынок приложений огромен, и хороший пользовательский интерфейс может выделить ваше приложение на фоне конкурентов.

Даже если пользовательский интерфейс спроектирован специалистом, вы должны реализовать его. В этой будут рассмотрены некоторые инструменты, помогающие быстро создать прототип дизайна приложения. Работая с профессиональным дизайнером, вы будете знать, чего от него требовать и как использовать созданные им ресурсы.

В этих двух главах мы создадим приложение, имитирующее телевизионный пульт дистанционного управления. Впрочем, ничем управлять оно не будет; это всего лишь тренажер для отработки навыков проектирования дизайна. В этой главе мы используем стили и включения для создания пульта, изображенного на рис. 24.1.

Приложение RemoteControl содержит только одну активность, изображенную на рис. 24.1. В верхней части пульта выводится текущий канал. Ниже находится область, в которой новый канал отображается в процессе ввода. Нажатие кнопки Delete очищает область ввода канала. Кнопка Enter изменяет канал с обновлением области текущего канала и очисткой области ввода.

Создание проекта RemoteControl

Создайте новый проект Android Application и настройте его параметры, как показано на рис. 24.2.

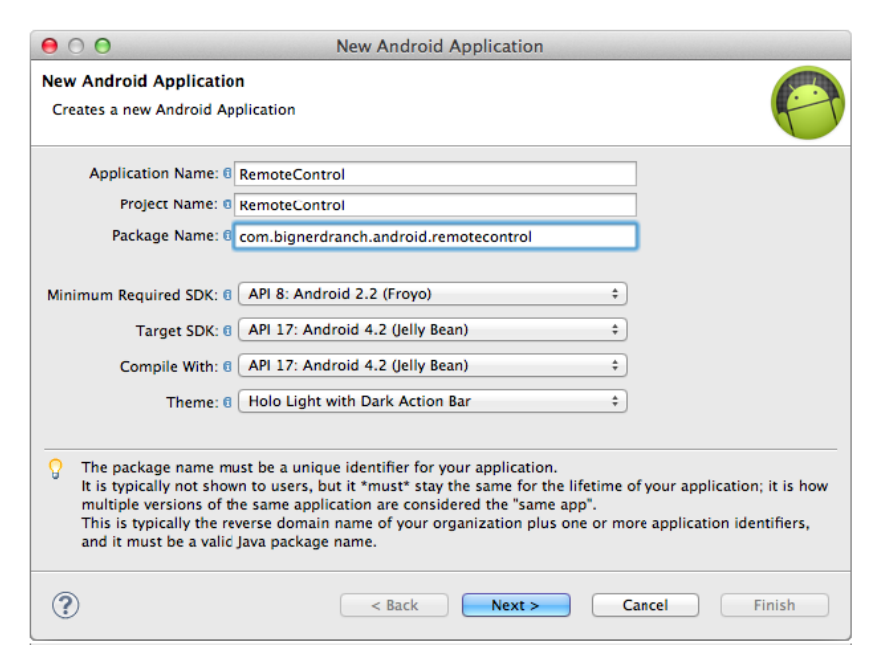


Рис. 24.2. Создание проекта RemoteControl

Прикажите мастеру создать пустую активность с именем `RemoteControlActivity`.

Создание RemoteControlActivity

Так как класс `RemoteControlActivity` будет субклассом `SingleFragmentActivity`, скопируйте файл SingleFragmentActivity.java из проекта CriminalIntent в пакет `com.bignerdranch.android.remotecontrol`. Затем скопируйте файл activity_fragment.xml в каталог res/layout проекта RemoteControl.

Откройте файл RemoteControlActivity.java. Назначьте класс `RemoteControlActivity` субклассом `SingleFragmentActivity`; он будет создавать экземпляр `RemoteControlFragment`. (Вскоре мы создадим этот класс фрагмента). Наконец, переопределите метод `RemoteControlActivity.onCreate(…)`, чтобы он скрывал панель действий или панель заголовка активности.

Листинг 24.1. Создание класса RemoteControlActivity (RemoteControlActivity.java)

```
public class RemoteControlActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

продолжение ⇗

Листинг 24.1 (*продолжение*)

```
        setContentView(R.layout.activity_remote_control);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_remote_control, menu);
        return true;
    }

    @Override
    protected Fragment createFragment() {
        return new RemoteControlFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        super.onCreate(savedInstanceState);
    }
}
```

Откройте файл AndroidManifest.xml и ограничьте эту активность портретной ориентацией.

Листинг 24.2. Ограничьтесь портретной ориентацией (AndroidManifest.xml)

```
<activity
      android:name="com.bignerdranch.android.remotecontrol.RemoteControlActivity"
  android:label="@string/app_name"
  android:screenOrientation="portrait">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Создание RemoteControlFragment

На панели Package Explorer переименуйте файл activity_remote_control.xml в fragment_remote_control.xml. Для начала мы создадим трехкнопочный пульт, чтобы упростить код. Замените содержимое fragment_remote_control.xml разметкой XML из листинга 24.3.

Листинг 24.3. Исходный трехкнопочный макет (layout/fragment_remote_control.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".RemoteControlActivity" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />

</RelativeLayout>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragment_remote_control_tableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:stretchColumns="*" >
  <TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:gravity="center"
    android:text="0"
    android:textSize="50dp" />
  <TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:layout_margin="15dp"
    android:background="#555555"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textSize="20dp" />
  <TableRow android:layout_weight="1" >
    <Button
      android:id="@+id/fragment_remote_control_zeroButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="0" />
    <Button
      android:id="@+id/fragment_remote_control_oneButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="1" />
    <Button
      android:id="@+id/fragment_remote_control_enterButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:text="Enter" />
  </TableRow>
</TableLayout>
```

Конструкция `android:stretchColumns="*"` гарантирует, что все столбцы будут иметь одинаковую ширину. Кроме того, для определения размера текста мы используем единицы `dp` вместо `sp`. Это означает, что текст будет иметь одинаковый размер независимо от настроек пользователя.

Наконец, создайте новый класс с именем `RemoteControlFragment`. Назначьте его суперклассом `android.support.v4.app.Fragment`. В файле RemoteControlFragment.java переопределите метод `onCreateView(…)` с назначением слушателей для кнопок.

Листинг 24.4. Создание класса RemoteControlFragment (RemoteControlFragment.java)

```
public class RemoteControlFragment extends Fragment {
    private TextView mSelectedTextView;
    private TextView mWorkingTextView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_remote_control, parent, false);

        mSelectedTextView = (TextView)v
                .findViewById(R.id.fragment_remote_control_selectedTextView);
        mWorkingTextView = (TextView)v
                .findViewById(R.id.fragment_remote_control_workingTextView);

        View.OnClickListener numberButtonListener = new View.OnClickListener() {
            public void onClick(View v) {
                TextView textView = (TextView)v;
                String working = mWorkingTextView.getText().toString();
                String text = textView.getText().toString();
                if (working.equals("0")) {
                    mWorkingTextView.setText(text);
                } else {
                    mWorkingTextView.setText(working + text);
                }
            }
        };

        Button zeroButton = (Button)v
            .findViewById(R.id.fragment_remote_control_zeroButton);
        zeroButton.setOnClickListener(numberButtonListener);

        Button oneButton = (Button)v
            .findViewById(R.id.fragment_remote_control_oneButton);
        oneButton.setOnClickListener(numberButtonListener);

        Button enterButton = (Button) v
            .findViewById(R.id.fragment_remote_control_enterButton);
        enterButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                CharSequence working = mWorkingTextView.getText();
                if (working.length() > 0)
                    mSelectedTextView.setText(working);
                mWorkingTextView.setText("0");
            }
        });

        return v;
    }
}
```

Хотя мы используем три кнопки, нужны только два слушателя щелчков. Дело в том, что для обоих цифровых кнопок может использоваться один слушатель. При щелчке на цифровой кнопке вы либо добавляете новую цифру в существующем текстовом представлении, либо заменяете его цифрой, на которой был сделан

щелчок — в зависимости от того, является ли текущим введенным текстом 0. Затем при нажатии кнопки Enter текст рабочей области перемещается в область выбора и стирается.

Запустите приложение RemoteControl. Вы увидите простое трехкнопочное приложение с двоичным вводом. Да если разобраться, кому нужно больше двух кнопок?

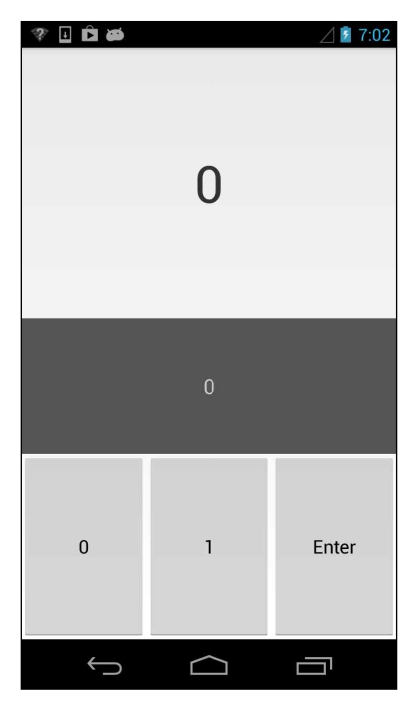


Рис. 24.3. Пульт с двоичным вводом

Стили и компактность разметки

Теперь, когда у нас имеется работоспособный проект, взгляните на разметку макета. Все кнопки выглядят одинаково. Пока это не создает проблем, но если вы захотите добавить к кнопке атрибут, работу придется повторить в трех местах. А что будет, когда кнопок станет двенадцать?

К счастью, Android поддерживает стили пользовательского интерфейса, которые позволяют избавиться от подобных повторений. Ресурсы стилей похожи на стили CSS. Каждый стиль определяет набор пар XML «атрибут-значение». Стили организуются в иерархию: потомок обладает теми же атрибутами и значениями, что и родитель, но может переопределять их или добавлять новые значения.

Стили, как и строковые ресурсы, определяются в теге `<resources>` файла XML из папки res/values. Имя файла, как и для строковых ресурсов, несущественно, но по общепринятой схеме стили размещаются в файле styles.xml.

Мастер проектов Android уже создал за вас файл styles.xml. (Обратите внимание: в этом файле уже определена тема приложения RemoteControl — стандартная для Android; тема определяет внешний вид кнопок, фоны и другие распространенные элементы.)

Мы переместим атрибуты, общие для всех кнопок, из конкретных кнопок в новый стиль RemoteButton. Включите в файл стилей определение, приведенное ниже.

Листинг 24.5. Исходные определения стилей RemoteControl (values/styles.xml)

```
<resources>

  <style name="AppTheme" parent="android:Theme.Light" />

  <style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
  </style>

</resources>
```

Каждый стиль определяется элементом <style> с одним или несколькими элементами <item>. У каждого элемента есть имя, которое соответствует имени атрибута XML, а текст внутри элемента определяет значение, применяемое к атрибуту.

В нашем примере используется всего один стиль с именем RemoteButton, который пока делает не так уж много. Впрочем, вскоре наша программа станет куда более «стильной».

Чтобы использовать стиль, укажите его в атрибуте style макета. Внесите изменения в файл fragment_remote_control.xml: удалите старые атрибуты и используйте вместо них новый стиль.

Листинг 24.6. Применение стилей к макету (layout/activity_remote.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ...
  >

  ...

  <TableRow android:layout_weight="1" >
    <Button
      android:id="@+id/fragment_remote_control_zeroButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      style="@style/RemoteButton"
      android:text="0" />
    <Button
      android:id="@+id/fragment_remote_control_oneButton"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      style="@style/RemoteButton"
      android:text="1" />
    <Button
```

```
        android:id="@+id/fragment_remote_control_enterButton"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        style="@style/RemoteButton"
        android:text="Enter" />
  </TableRow>
</TableLayout>
```

Запустите приложение RemoteControl. Внешний вид пульта нисколько не изменился, но разметка стала более компактной, а количество повторений в ней уменьшилось.

Завершение разметки

Готовые стили способны значительно ускорить разработку. Давайте превратим интерфейс пульта в полноценную роскошную версию с 12 кнопками, с поддержкой десяти цифр.

Для этого нужно создать массив 3 × 4 практически идентичных кнопок. Но определять их все по отдельности не обязательно — можно создать одну строку из трех кнопок и использовать ее четыре раза. Создайте новый файл с именем res/layout/button_row.xml и переместите в него уже определенную строку кнопок.

Листинг 24.7. Строка из трех кнопок (layout/button_row.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
  <Button style="@style/RemoteButton" />
  <Button style="@style/RemoteButton" />
  <Button style="@style/RemoteButton" />
</TableRow>
```

Остается «включить» эту строку в разметку четыре раза. Как это сделать? При помощи тега `include`.

Листинг 24.8. Включение строк кнопок в основной макет (layout/fragment_remote_control.xml)

```
<TableRow android:layout_weight="1" >
  <Button
    android:id="@+id/fragment_remote_control_zeroButton"
    style="@style/RemoteButton"
    android:text="0" />
  <Button
    android:id="@+id/fragment_remote_control_oneButton"
    style="@style/RemoteButton"
    android:text="1" />
  <Button
    android:id="@+id/fragment_remote_control_enterButton"
    style="@style/RemoteButton"
    android:text="Enter" />
</TableRow>
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
```

продолжение ⇗

Листинг 24.8 (*продолжение*)

```
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
<include
  android:layout_weight="1"
  layout="@layout/button_row" />
<include
  android:layout_weight="1"
    layout="@layout/button_row" />
```

Вероятно, вы заметили, что три кнопки, определенные в layout/button_row.xml, не имеют идентификаторов. Поскольку строка кнопок включается четыре раза, если бы мы назначили идентификаторы кнопкам, они были бы не уникальными. По этой же причине для кнопок не определяется текст — это нормально, все присваивания можно выполнить в коде. Давайте подключим кнопки и зададим их текст в коде.

Листинг 24.9. Подключение цифровых кнопок (RemoteControlFragment.java)

```
View.OnClickListener numberButtonListener = new View.OnClickListener() {
  ...
};

Button zeroButton = (Button)v.findViewById(R.id.fragment_remote_control_zeroButton);
zeroButton.setOnClickListener(numberButtonListener);

Button oneButton = (Button)v.findViewById(R.id.fragment_remote_control_oneButton);
oneButton.setOnClickListener(numberButtonListener);
TableLayout tableLayout = (TableLayout)v
        .findViewById(R.id.fragment_remote_control_tableLayout);
int number = 1;
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    TableRow row = (TableRow)tableLayout.getChildAt(i);
    for (int j = 0; j < row.getChildCount(); j++) {
        Button button = (Button)row.getChildAt(j);
        button.setText("" + number);
        button.setOnClickListener(numberButtonListener);
        number++;
    }
}
```

Цикл `for` начинается с индекса 2, чтобы пропустить два текстовых представления, а затем перебирает все кнопки первых трех строк. Каждой кнопке назначается тот же слушатель `numberButtonListener`, который был создан ранее, и текст с соответствующей цифрой.

С тремя строками кнопок мы разобрались. С последней строкой все не так просто: нужно обработать особые случаи кнопок Delete и Enter.

Листинг 24.10. Подключение кнопок последней строки (RemoteControlFragment.java)

```
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    ...
}

TableRow bottomRow = (TableRow)tableLayout
        .getChildAt(tableLayout.getChildCount() - 1);
```

```
Button deleteButton = (Button)bottomRow.getChildAt(0);
deleteButton.setText("Delete");
deleteButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mWorkingTextView.setText("0");
    }
});

Button zeroButton = (Button)bottomRow.getChildAt(1);
zeroButton.setText("0");
zeroButton.setOnClickListener(numberButtonListener);

Button enterButton = (Button)
        v.findViewById(R.id.fragment_remote_control_enterButton);
Button enterButton = (Button)bottomRow.getChildAt(2);
enterButton.setText("Enter");
enterButton.setOnClickListener(new View.OnClickListener() {
  ...
});
```

Запустите приложение и поэкспериментируйте с ним. Пульт полностью функционален, хотя и ничем не управляет. Подключение кнопок к беспроводной связи с телевизором остается читателю для самостоятельной работы.

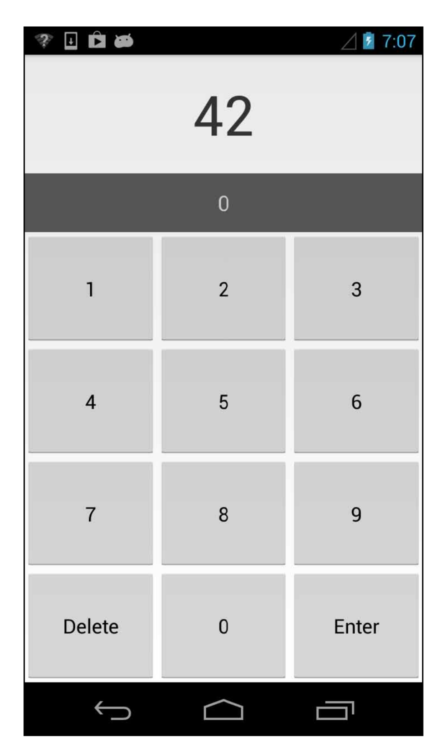


Рис. 24.4. Канал о смысле жизни, вселенной и всем таком

Внешний вид целой группы кнопок, к которым применены стили, можно изменить одним махом. Добавьте следующие три строки в стиль `RemoteButton`.

Листинг 24.11. Дополнительная настройка стиля (values/styles.xml)

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textColor">#556699</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
</style>
```

Взмах волшебной палочки — и все кнопки моментально изменяются!

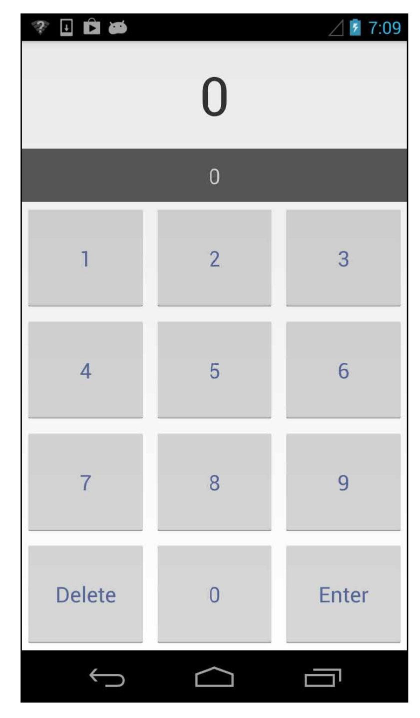


Рис. 24.5. Все кнопки выглядят одинаково!

Для любознательных: include и merge

Ранее в этой главе мы использовали тег `include` для многократного включения строки кнопок `RemoteButton` (в один макет). Принцип работы тега достаточно очевиден:

```
<include layout="@layout/some_partial_layout"/>
```

Этот элемент включает содержимое файла разметки с идентификатором ресурса `@layout/some_partial_layout`.

Тег `include` может использоваться как для устранения дублирования разметки в одном макете, как это было сделано ранее в этой главе, так и для устранения дублирования между макетами. Если некоторый блок разметки встречается в двух

и более макетах, вы можете создать его в форме макета и включать там, где он нужен. Если в будущем этот блок потребуется изменить, все изменения достаточно внести в одном месте.

У тегов `include` есть пара особенностей, о которых необходимо знать. Во-первых, включаемый макет проходит обычный процесс поиска ресурсов в зависимости от текущей конфигурации устройства; соответственно в нем можно использовать конфигурационные квалификаторы, как и в любом другом макете. Во-вторых, можно переопределять атрибут `android:id` и любые атрибуты `android:layout_*` корневого элемента включаемого макета, указывая их в самом теге `include`. Это позволяет использовать один макет несколько раз, указывая разные атрибуты при каждом включении.

Элемент `merge` работает в сочетании с элементом `include`. Он может использоваться как корневой элемент включаемого макета вместо реального виджета. Когда один макет включает другой макет с корневым элементом `merge`, потомки элемента `merge` включаются напрямую — они становятся потомками элемента `include`, а элемент `merge` удаляется.

Механизм включения работает проще, чем кажется из описания. Если оно вам кажется запутанным, просто запомните, что элемент `merge` отбрасывается. Он присутствует только по одной причине: XML требует, чтобы корневой элемент присутствовал только в одном экземпляре.

Упражнение. Наследование стилей

Кнопки Delete и Enter скромно располагаются у нижнего края экрана, а в их оформлении используется такой же стиль, как у цифровых кнопок. Кнопкам «действий» нужен другой, более броский стиль.

Создайте новый стиль кнопки, производный от стиля `RemoteButton`, который устанавливает атрибут полужирного текста. Затем включите использование нового стиля для первой и третьей кнопки в нижней строке.

Создать стиль, производный от другого стиля, несложно. Это можно сделать двумя способами. Первый — присвоить атрибуту `parent` вашего стиля имя того стиля, от которого он должен наследовать. Другой, более простой способ — снабдить имя стиля префиксом из родительского стиля и точки (например, `ParentStyleName.MyStyleName`).

25 Графические объекты

В предыдущей главе мы быстро построили интерфейс пульта дистанционного управления, используя некоторые нетривиальные приемы создания макетов. Пульт выглядит неплохо, хотя и немного банально, потому что все кнопки в нем имеют типичный для Android внешний вид и поведение. В этой главе мы воспользуемся двумя новыми инструментами, чтобы придать кнопкам совершенно особое оформление.

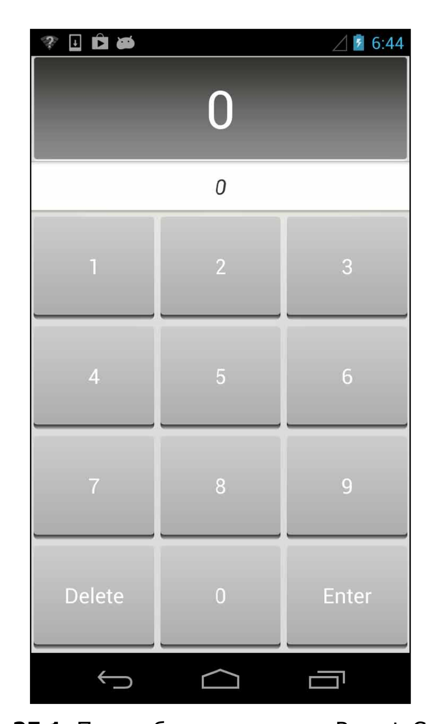


Рис. 25.1. Переработанная версия RemoteControl

Оба эти инструмента относятся к категории *графических объектов* (drawables). В Android графическим объектом называется все, что предназначено для прорисовки на

экране, будь то абстрактная фигура, класс, производный от `Drawable`, или растровое изображение. Нам уже знаком один вид графических объектов: `BitmapDrawable`, представляющий растровое изображение.

В этой главе мы рассмотрим несколько других видов графических объектов: списки состояний, геометрические фигуры, списки слоев и 9-зонные графические объекты. Поскольку первые три вида обычно определяются в файлах XML, мы объединим их в более широкую категорию *графических объектов XML*.

Графические объекты XML

Прежде чем браться за графические объекты XML, проведем небольшой эксперимент: используем атрибут `android:background` для изменения цвета фона кнопок.

Листинг 25.1. Попытка изменения цвета кнопки (values/styles.xml)

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textColor">#556699</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
  <item name="android:background">#ccd7ee</item>
</style>
```

Результат должен выглядеть примерно так.

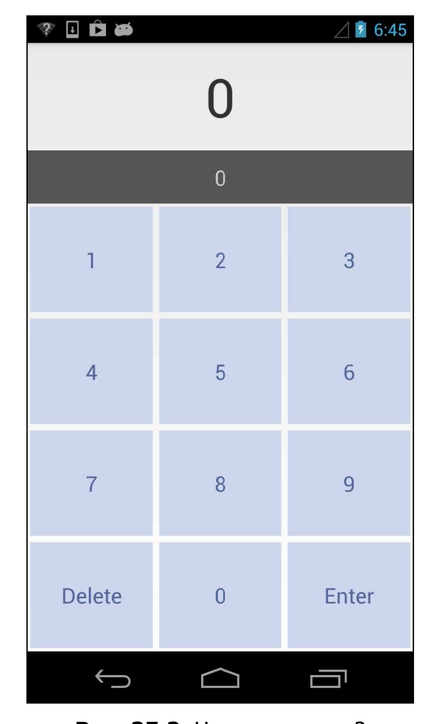


Рис. 25.2. Что произошло?

Трехмерное оформление кнопок исчезло. Также вы увидите, что внешний вид кнопок перестал изменяться при нажатии.

Почему все это произошло из-за одного безобидного изменения атрибута? Класс Button представляет собой не что иное, как View с применением стиля по умолчанию. Стиль берется из выбранной темы, а в качестве фона назначается объект Drawable. Он отвечает за имитацию объема и изменения внешнего вида при нажатии. К концу этой главы вы будете знать, как наделить такими функциями ваши графические объекты.

Начнем с создания цветных геометрических фигур в XML. Поскольку графические объекты XML не привязаны к конкретной плотности пикселов, обычно они размещаются в папке drawable по умолчанию (а не в одной из специализированных папок).

На панели Package Explorer создайте каталог res/drawable. В этом каталоге создайте файл XML с именем button_shape_normal.xml. Назначьте его корневым элементом shape. (Почему кнопка названа «normal», то есть «нормальной»? Потому что скоро появится другая, менее нормальная.)

Листинг 25.2. Создание кнопки на базе геометрической фигуры (drawable/button_shape_normal.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >

  <corners android:radius="3dp" />

  <gradient
    android:angle="90"
    android:endColor="#cccccc"
    android:startColor="#acacac" />

</shape>
```

Этот файл определяет прямоугольник с закругленными углами. Элемент corners задает радиус закругления, а элементы gradient задают направление и начальный/конечный цвета градиента.

Вы можете использовать элемент shape для создания других фигур: эллипсов, линий, колец и т. д., а также назначать им разное оформление. За подробностями обращайтесь к документации по адресу http://developer.android.com/guide/topics/resources/drawable-resource.html.

В файле styles.xml обновите стиль Button, чтобы он использовал новый объект Drawable в качестве фона.

Листинг 25.3. Обновление стиля кнопки (values/styles.xml)

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textColor">#556699</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
```

```
  <item name="android:background">#ccd7ee</item>
  <item name="android:background">@drawable/button_shape_normal</item>
</style>
```

Запустите приложение RemoteControl и посмотрите, что изменилось.

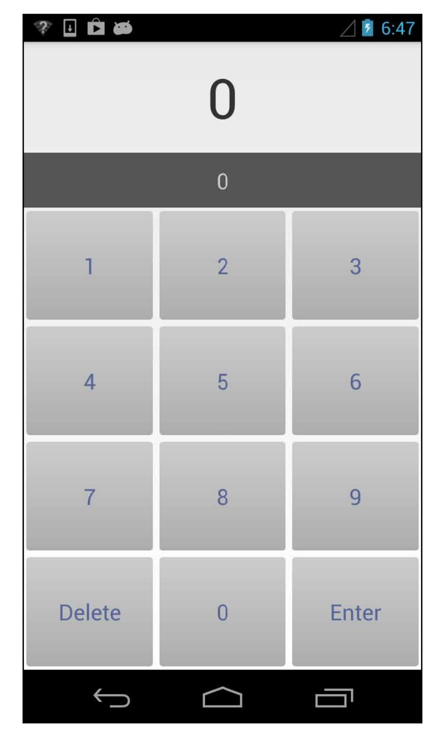


Рис. 25.3. Теперь с закругленными углами

Списки состояний

Эти кнопки смотрятся хорошо, но по сути это простые статичные изображения. Во внутренней реализации старый фон `Button` использовал графический объект *списка состояний* (state list). Списки состояний позволяют выводить разные графические объекты для каждого состояния, в котором находится ассоциированный объект `View`. (Ранее мы использовали список состояний для изменения фона элементов списка в главе 18.) Существует много разных состояний, но в данном случае нас интересует лишь то, нажата кнопка или нет.

Начнем с создания внешнего вида нажатой кнопки. Кнопка будет выглядеть так же, как обычная кнопка, отличаясь от нее только цветом.

На панели Package Explorer создайте копию button_shape_normal.xml и присвойте ей имя button_shape_pressed.xml. Откройте файл button_shape_pressed.xml и увеличьте угол на 180 градусов, чтобы изменить направление градиента.

Листинг 25.4. Создание фигуры для нажатой кнопки (drawable/button_shape_pressed.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >

  <corners android:radius="3dp" />

  <gradient
    android:angle="90"
    android:angle="270"
    android:endColor="#cccccc"
    android:startColor="#acacac" />

</shape>
```

Затем нам понадобится графический объект списка состояний. Графический объект списка состояний должен иметь корневой элемент `selector` и один или несколько элементов `item`, описывающих разные состояния. Щелкните правой кнопкой мыши на каталоге res/drawable/, создайте новый файл XML с именем button_shape.xml и назначьте ему корневой элемент `selector`.

Листинг 25.5. Создание фигуры для интерактивной кнопки (drawable/button_shape.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/button_shape_normal"
    android:state_pressed="false"/>
  <item android:drawable="@drawable/button_shape_pressed"
    android:state_pressed="true"/>
</selector>
```

Ненажатая кнопка имеет темный текст, который хорошо смотрится на светлом фоне. Теперь, когда фон стал более темным, в ненажатом состоянии светлый цвет будет смотреться лучше, а темный цвет можно использовать для фона в нажатом состоянии. Списки состояний для цветов создаются так же, как списки состояний для фигур, так что задача решается просто.

Щелкните правой кнопкой мыши на каталоге res/drawable/ и создайте другой графический объект списка состояний с именем button_text_color.xml.

Листинг 25.6. Изменение цвета текста в зависимости от состояния (drawable/button_text_color.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="false" android:color="#ffffff"/>
    <item android:state_pressed="true" android:color="#556699"/>
</selector>
```

В файле styles.xml измените стиль `Button`, включите в него ссылки на новый графический объект фона и новый цвет текста.

Листинг 25.7. Обновление стиля кнопки (values/styles.xml)

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textColor">#556699</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
  <item name="android:background">@drawable/button_shape_normal</item>
  <item name="android:background">@drawable/button_shape</item>
  <item name="android:textColor">@drawable/button_text_color</item>
</style>
```

Запустите приложение RemoteControl. Посмотрите, как выглядит нажатая кнопка.

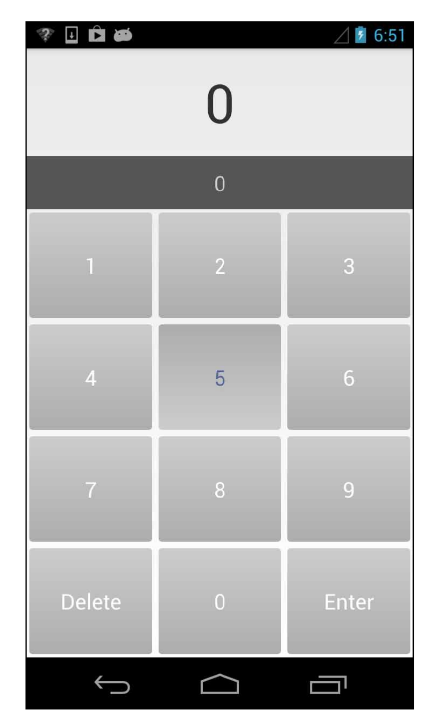


Рис. 25.4. Нажатая кнопка

Списки слоев и смещение

Стандартные кнопки Android, которые вы видели в исходной версии приложения, имитируют эффект тени. К сожалению, у фигур нет свойства тени, однако вы можете реализовать эффект тени самостоятельно при помощи двух других разновидностей графических объектов XML: *списков слоев* и *смещений*.

Сначала создается тень такой же формы, как у текущей кнопки. Она объединяется с текущей кнопкой с использованием тега `layer-list`, а нижний край кнопки немного смещается с использованием тега `inset`, чтобы стала видна находящаяся внизу тень.

Создайте новый файл XML в каталоге res/drawable/. Присвойте ему имя button_shape_shadowed.xml и назначьте корневым элементом `layer-list`.

Листинг 25.8. Имитация тени у кнопок (drawable/button_shape_shadowed.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
  <item>
    <shape android:shape="rectangle" >
      <corners android:radius="5dp" />

      <gradient
        android:angle="90"
        android:centerColor="#303339"
        android:centerY="0.05"
        android:endColor="#000000"
        android:startColor="#00000000" />
    </shape>
  </item>
  <item>
    <inset
      android:drawable="@drawable/button_shape"
      android:insetBottom="5dp" />
  </item>
</layer-list>
```

Список слоев содержит несколько графических объектов, упорядоченных в порядке их прорисовки на экране (от заднего плана к переднему). Вторым графическим объектом в списке является `inset`, который всего лишь смещает нижний край уже созданного графического объекта на 5dp, чтобы он находился выше созданной тени.

Также обратите внимание на то, что вместо использования отдельных файлов для графического объекта тени мы встраиваем его непосредственно в список слоев. Этот способ также подходит и для других графических объектов — как, например, упоминавшихся выше списков состояний. Вкладывать ли графические объекты или выделять их в отдельные файлы, как было сделано ранее, — решайте сами. Выделение сокращает дублирование кода и упрощает файлы, но приводит к загромождению каталога drawable/folder дополнительными файлами. Используйте код, который покажется вам наиболее понятным и удобочитаемым.

В файле styles.xml включите в стиль кнопки ссылку на новый графический объект с тенью.

Листинг 25.9. Последнее изменение стиля кнопки (values/styles.xml)

```
<style name="RemoteButton">
  <item name="android:layout_width">0dp</item>
  <item name="android:layout_height">match_parent</item>
  <item name="android:textSize">20dp</item>
  <item name="android:layout_margin">3dp</item>
  <item name="android:background">@drawable/button_shape_normal</item>
```

```
  <item name="android:background">@drawable/button_shape_shadowed</item>
  <item name="android:textColor">@drawable/button_text_color</item>
</style>
```

И последнее изменение: создадим градиентный графический объект для всего главного представления, чтобы имитировать эффект освещения.

Создайте новый файл с именем remote_background.xml и корневым элементом shape. Добавьте в него следующий блок разметки (примечание: если фигура не указана, по умолчанию используется прямоугольник).

Листинг 25.10. Новый фоновый графический объект для корневого представления (drawable/remote_background.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
  <gradient
    android:centerY="0.05"
    android:endColor="#dbdbdb"
    android:gradientRadius="500"
    android:startColor="#f4f4e9"
    android:type="radial" />
</shape>
```

Добавьте новый графический объект в файл fragment_remote_control.xml.

Листинг 25.11. Включение графического объекта в макет (layout/fragment_remote_control.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/fragment_remote_control_tableLayout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="@drawable/remote_background"
  android:stretchColumns="*" >
```

При внесении всех этих изменений мы ничего не делали с файлом макета фрагмента. (Не считая последнего изменения, и конечно, мы могли создать новый стиль для TableLayout.) Это хорошо, поскольку мы не изменяем содержимое макета, а только его презентацию. Добавление новых кнопок или перемещения потребуют обновления файла макета, но изменения во внешнем виде можно вносить на уровне стиля.

9-зонные изображения

Если вы привлечете к построению пользовательского интерфейса профессионального дизайнера, результат его работы не всегда может быть реализован графическими объектами XML. Тем не менее готовые изображения иногда удается использовать в разных местах. Для «растяжимых» (stretchable) элементов пользовательского интерфейса — например, для фонов кнопок — в Android существуют так называемые *9-зонные изображения* (9-patch).

Наша следующая задача будет связана с двумя виджетами TextView в верхней части экрана. Макет кнопок останется неизменным, но для фонов будут использоваться

растяжимые графические объекты. Оба объекта существенно уже `TextView`, так что они не займут много места. Первый — растяжимый фон window.png — придает `TextView` изящный цветовой переход, а также создает обрамление по краям.

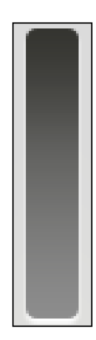


Рис. 25.5. Область для вывода канала

Другой объект — bar.png — создает небольшую тень внизу и тонкую рамку наверху. Следующий рисунок заметно увеличен; изображение довольно маленькое.

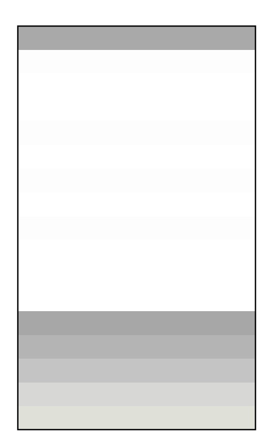


Рис. 25.6. Область для ввода канала

Вы найдете эти изображения в архиве решений, в каталоге 25_XMLDrawables/RemoteControl/res/drawable-hdpi. Скопируйте их в папку /res/drawable-hdpi проекта RemoteControl.

Отредактируйте файл fragment_remote_control.xml и назначьте эти изображения фонами для соответствующих представлений. Также выберите цвет текста, более подходящий для новых фонов, — белый для верхней области, черный (используемый по умолчанию) для нижней. Затем включите для нижнего виджета `TextView` курсивное начертание, с которым текст будет лучше смотреться.

Листинг 25.12. Добавление графических объектов в стили (layout/fragment_remote_control.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  <TextView
```

```
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:background="@drawable/window"
    android:gravity="center"
    android:text="0"
    android:textColor="#ffffff"
    android:textSize="50dp" />
  <TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_margin="15dp"
    android:layout_weight="1"
    android:background="#555555"
    android:background="@drawable/bar"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textStyle="italic"
    android:textSize="20dp" />
  ...
</TableLayout>
```

А теперь посмотрите, что происходит при запуске приложения.

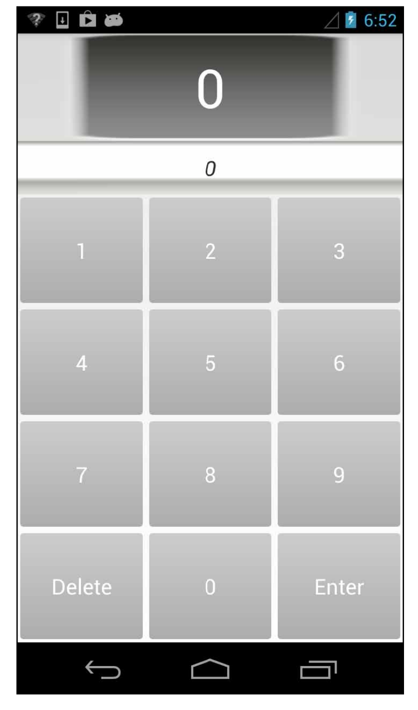


Рис. 25.7. Разбитые мечты

Каждое изображение было равномерно растянуто по всем направлениям для заполнения представления. В некоторых ситуациях это именно то, что нужно — но не в нашем случае. Изображения размываются, а у цифр в нижнем виджете TextView нарушается выравнивание по центру.

Для решения этой проблемы можно использовать 9-зонные изображения. Файл 9-зонного изображения специально форматируется так, чтобы система Android знала, какие части изображения можно или нельзя форматировать. При правильной реализации это гарантирует, что края и углы фона будут соответствовать изображению в том виде, в котором оно было создано.

Почему изображения называются «9-зонными»? Такие изображения разбиваются сеткой 3 × 3 — такая сетка состоит из 9 элементов, или *зон* (patches). Углы сетки не масштабируются, стороны масштабируются только в одном направлении, а центральная зона масштабируется в обоих направлениях.

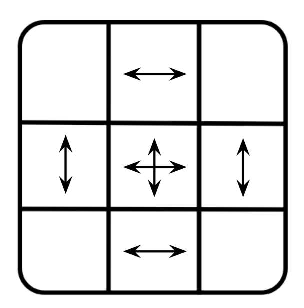


Рис. 25.8. Как работает 9-зонное изображение

9-зонное изображение во всех отношениях напоминает обычный файл png, за исключением двух аспектов: имя файла завершается суффиксом .9.png, и изображение имеет дополнительную рамку толщиной в один пиксел. Рамка задает местонахождение центрального квадрата. Черными пикселами рамки обозначается центр, а прозрачными — края.

9-зонное изображение можно создать в любом графическом редакторе, но проще воспользоваться программой draw9patch, включенной в поставку Android SDK. Программа находится в каталоге tools установки SDK. Когда программа будет запущена, вы можете перетащить на нее нужный файл или открыть его из меню File.

Когда файл будет открыт, заполните черные пикселы на верхней и левой границе, чтобы обозначить растягиваемые области изображения. Добавьте пикселы в window.png, как показано на рис. 25.9.

Левый и верхний участки границы отмечают растягиваемую область изображения. А как насчет правого и нижнего участка? Они отмечают необязательную *область прорисовки* — область, в которой должно выводиться некое содержимое (обычно текст). Если область прорисовки не задана, по умолчанию считается, что она совпадает с растягиваемой областью. В нашем случае это именно так, поэтому область прорисовки не указывается.

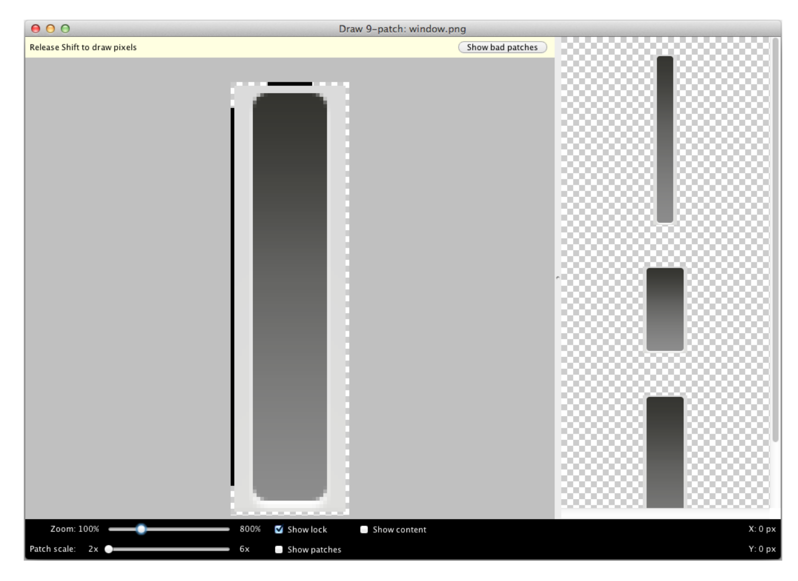


Рис. 25.9. 9-зонное изображение для отображения канала

Когда работа будет завершена, сохраните результаты в файле с именем window_patch.9.png. Будьте внимательны и следите за тем, чтобы имена 9-зонных файлов не конфликтовали с файлами других изображений; если у вас имеются файлы с именами window.9.png и window.png, приложение не построится.

Затем отредактируйте 9-зонное изображение bar.png. Используйте область прорисовки для центровки текста по вертикали и горизонтали, а также создания со всех сторон полей толщиной 4 пиксела.

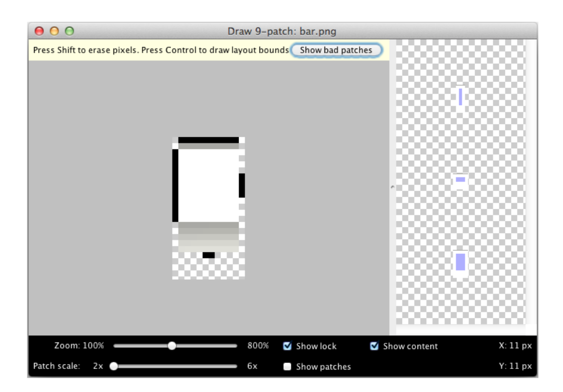


Рис. 25.10. 9-зонное изображение для области вывода канала

Сохраните готовое изображение в файле bar_patch.9.png.

Щелкните правой кнопкой мыши на каталоге res/ и выберите команду Refresh. Eclipse живет в своем обособленном мире, а команда Refresh напоминает ему о необходимости снова свериться с файловой системой.

Наконец, переключите две области `TextView` на использование 9-зонных изображений вместо обычных.

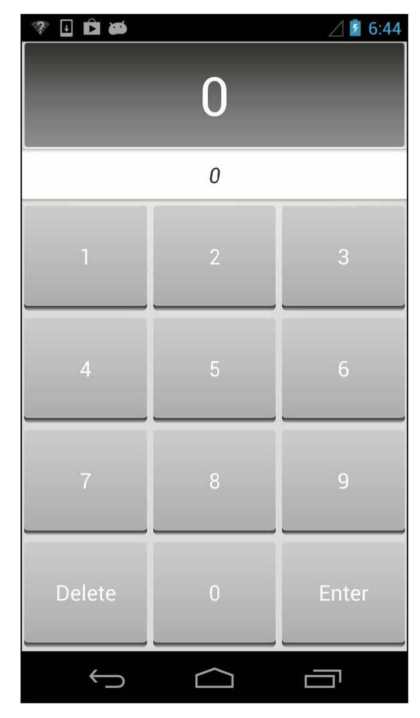


Рис. 25.11. Переработанная версия RemoteControl

Листинг 25.13. Переход на 9-зонные изображения (layout/fragment_remote_control.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  <TextView
    android:id="@+id/fragment_remote_control_selectedTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:background="@drawable/window"
    android:background="@drawable/window_patch"
    android:gravity="center"
    android:text="0"
    android:textSize="50dp" />
  <TextView
    android:id="@+id/fragment_remote_control_workingTextView"
    android:layout_width="match_parent"
    android:layout_height="0dp"
```

```
    android:layout_margin="15dp"
    android:layout_weight="1"
    android:background="@drawable/bar"
    android:background="@drawable/bar_patch"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textSize="20dp" />
  ...
</TableLayout>
```

Запустите приложение. Вуаля! В нем появились работающие фоновые изображения. Помните, как банально выглядел исходный интерфейс? Чтобы создать нечто куда более интересное, нам потребовалось совсем немного времени. Представьте, чего можно добиться с помощью профессионального дизайнера. Сделайте ваше приложение приятным для глаза, удобным в использовании — и ваши усилия окупятся его популярностью.

26 HTTP и фоновые задачи

В умах пользователей господствуют сетевые приложения. Чем заняты эти люди, которые за обедом возятся со своими телефонами вместо дружеской беседы? Они маниакально проверяют поставки новостей, отвечают на текстовые сообщения или играют в сетевые игры.

Для экспериментов с сетевыми возможностями Android мы создадим новое приложение PhotoGallery. Это клиент для сайта фотообмена Flickr, который будет загружать и отображать последние общедоступные фото, отправленные на Flickr. Рисунок 26.1 дает примерное представление о внешнем виде приложения.

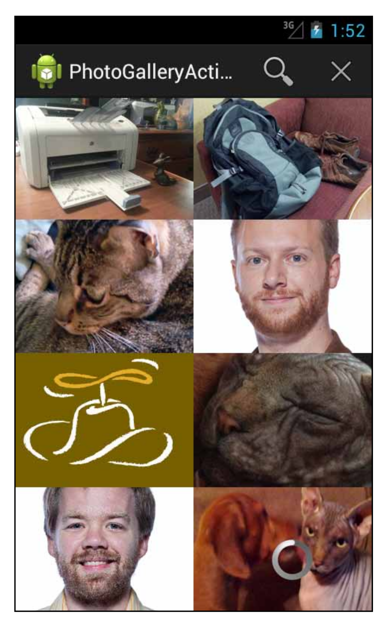


Рис. 26.1. Приложение PhotoGallery

(На рис. 26.1 изображены наши фотографии вместо общедоступных фотографий с Flickr. Фотографии на сайте Flickr являются собственностью человека, отправившего их, и не могут повторно использоваться без разрешения владельца. За дополнительной информацией об использовании независимого контента, загруженного с Flickr, обращайтесь по адресу *http://pressroom.yahoo.net/pr/ycorp/permissions.aspx*.)

За созданием приложения PhotoGallery мы проведем шесть глав. Две главы будут посвящены основам загрузки и разбора XML, а также выводу изображений. В последующих главах будут реализованы дополнительные функции: поиск, сервисы, оповещения, получатели широковещательных рассылок и веб-представления.

В этой главе вы научитесь использовать высокоуровневые сетевые средства HTTP в Android. Почти все программирование веб-служб в наши дни базируется на сетевом протоколе HTTP. К концу этой главы наше приложение будет загружать с Flickr, разбирать и отображать названия фотографий (загрузка и отображение самих фотографий откладывается до главы 27).



Рис. 26.2. PhotoGallery в конце этой главы

Создание приложения PhotoGallery

Создайте новый проект приложения Android. Задайте его параметры так, как показано на рис. 26.3.

Рис. 26.3. Создание приложения PhotoGallery

Прикажите мастеру создать пустую активность с именем `PhotoGalleryActivity`.

В приложении PhotoGallery используется такая же архитектура, как во всех предыдущих приложениях. Активность `PhotoGalleryActivity` будет субклассом `SingleFragmentActivity`, а ее представлением будет контейнерное представление, определяемое в файле activity_fragment.xml. Эта активность станет хостом фрагмента, а именно экземпляра `PhotoGalleryFragment`, который мы вскоре создадим.

Скопируйте файлы SingleFragmentActivity.java и activity_fragment.xml в свой проект из предыдущего проекта.

В файле PhotoGalleryActivity.java настройте `PhotoGalleryActivity` как `SingleFragmentActivity`; для этого удалите код, сгенерированный шаблоном, и замените его реализацией `createFragment()`. Метод `createFragment()` должен возвращать экземпляр `PhotoGalleryFragment`.

(Пока не обращайте внимания на ошибку, которая будет обнаружена в этом коде. Она исчезнет после того, как вы создадите класс `PhotoGalleryFragment`.)

Листинг 26.1. Настройка активности (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends Activity {
public class PhotoGalleryActivity extends SingleFragmentActivity {

    /* Код, сгенерированный шаблоном */

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }
}
```

PhotoGallery будет отображать свои результаты в виджете GridView, который станет представлением для PhotoGalleryFragment.

Класс GridView происходит от AdapterView, поэтому он работает по тем же принципам, что и ListView. Однако в отличие от ListView, GridView не имеет удобного класса GridFragment, который подключит все за вас. А это означает, что вам придется создать файл макета и заполнить его в PhotoGalleryFragment. Позднее в этой главе мы подключим в PhotoGalleryFragment адаптер, который будет поставлять GridView названия отображаемых фотографий.

Чтобы создать макет фрагмента, переименуйте файл layout/activity_photo_gallery.xml в layout/fragment_photo_gallery.xml. Затем замените его содержимое определением GridView, приведенным на рис. 26.4.

GridView

xmlns:android="**http://schemas.android.com/apk/res/android**"
xmlns:tools="**http://schemas.android.com/tools**"
android:id="**@+id/gridView**"
android:layout_width="**match_parent**"
android:layout_height="**match_parent**"
android:columnWidth="**120dp**"
android:numColumns="**auto_fit**"
android:stretchMode="**columnWidth**"

Рис. 26.4. GridView (layout/fragment_photo_gallery.xml)

Здесь мы назначаем столбцам ширину 120dp и приказываем GridView создать столько столбцов, сколько поместится на экране. Если в выделенной области осталось место размером менее 120dp, атрибут stretchMode приказывает GridView равномерно распределить его между столбцами.

Наконец, создайте класс PhotoGalleryFragment. Сохраните фрагмент, заполните созданный макет и получите ссылку на GridView (листинг 26.2).

Листинг 26.2. Заготовка кода (PhotoGalleryFragment.java)

```
package com.bignerdranch.android.photogallery;
...
public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
```

продолжение ⇨

Листинг 26.2 (*продолжение*)

```
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container,
false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        return v;
    }
}
```

Прежде чем переходить к следующему разделу, запустите приложение PhotoGallery и убедитесь в том, что все работает правильно (то есть если вы стали гордым владельцем пустого экрана).

Основы сетевой поддержки

В нашем приложении все сетевые взаимодействия PhotoGallery будут обеспечиваться одним классом. Создайте новый класс Java. Поскольку мы будем подключаться к Flickr, назовите класс `FlickrFetchr`.

Исходная версия `FlickrFetchr` будет состоять всего из двух методов: `getUrlBytes(String)` и `getUrl(String)`. Метод `getUrlBytes(String)` получает низкоуровневые данные по URL и возвращает их в виде массива байтов. Метод `getUrl(String)` преобразует результат из `getUrlBytes(String)` в `String`.

Добавьте в файл FlickrFetchr.java реализации `getUrlBytes(String)` и `getUrl(String)` (листинг 26.3).

Листинг 26.3. Основной сетевой код (FlickrFetchr.java)

```
package com.bignerdranch.android.photogallery;
...
public class FlickrFetchr {
    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();
            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return null;
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }
```

```
    public String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

Этот код создает объект URL на базе строки — например, *http://www.google.com*. Затем вызов метода openConnection() создает объект подключения к заданному URL-адресу. Вызов URL.openConnection() возвращает URLConnection, но поскольку подключение осуществляется по протоколу HTTP, мы можем преобразовать его в HttpURLConnection. Это открывает доступ к HTTP-интерфейсам для работы с методами запросов, кодами ответов, методами потоковой передачи и т. д.

Объект HttpURLConnection представляет подключение, но связь с конечной точкой будет установлена только после вызова getInputStream() (или getOutputStream() для POST-вызовов). До этого момента вы не сможете получить действительный код ответа.

После создания объекта URL и открытия подключения программа многократно вызывает read(), пока в подключении не кончатся данные. Объект InputStream предоставляет байты по мере их доступности. Когда чтение будет завершено, программа закрывает его и выдает массив байтов из ByteArrayOutputStream.

Хотя всю основную работу выполняет метод getUrlBytes(String), в этой главе мы будем использовать метод getUrl(String). Он преобразует байты, полученные вызовом getUrlBytes(String), в String. На данный момент это решение смотрится немного странно — зачем разбивать выполняемую работу на два метода? Однако наличие двух методов будет полезно в следующей главе, когда мы займемся загрузкой данных изображений.

Разрешение на работу с сетью

Для работы сетевой поддержки необходимо сделать еще одно: вы должны попросить разрешения. Никому из пользователей не понравится, если вы будете тайком загружать их фотографии.

Чтобы запросить разрешение на работу с сетью, добавьте следующую строку в файл AndroidManifest.xml.

Листинг 26.4. Включение разрешения на работу с сетью в манифест (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET" />

  ...
</manifest>
```

Использование AsyncTask для выполнения в фоновом потоке

На следующем шаге мы должны вызвать и протестировать только что добавленный сетевой код. Однако мы не можем просто вызвать FlickrFetchr.getURL(String) прямо из PhotoGalleryFragment. Вместо этого необходимо создать фоновый программный поток и выполнить код в нем.

Для работы с фоновыми потоками проще всего использовать вспомогательный класс с именем AsyncTask. AsyncTask создает фоновый поток и выполняет в нем код, содержащийся в методе doInBackground(…).

В файле PhotoGalleryFragment.java добавьте в конце PhotoGalleryFragment новый внутренний класс с именем FetchItemsTask. Переопределите метод AsyncTask.doInBackground(…) для получения данных с сайта и их регистрации в журнале. Затем используйте новый класс внутри PhotoGalleryFragment.onCreate(…).

Листинг 26.5. Реализация AsyncTask (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ...

    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr().getUrl("http://www.google.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

Затем в методе PhotoGalleryFragment.onCreate(…) вызовите метод execute() для нового экземпляра FetchItemsTask.

Листинг 26.6. Реализация AsyncTask (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
```

```
        new FetchItemsTask().execute();
    }
    ...
}
```

Вызов `execute()` активизирует класс `AsyncTask`, который запускает свой фоновый поток и вызывает `doInBackground(…)`. Выполните свой код и вы увидите, что в LogCat появляется разметка HTML домашней страницы Google Javascriptlicious — примерно так, как показано на рис. 26.5.

Рис. 26.5. Разметка HTML от Google в LogCat

Теперь, когда мы создали фоновый поток и выполнили в нем сетевой код, давайте поближе познакомимся с программным потоками в Android.

Главный программный поток

Сетевые взаимодействия не происходят моментально. Веб-серверу может потребоваться одна-две секунды на ответ, а загрузка файла может занять еще больше времени. Из-за продолжительности сетевых операций Android запрещает их выполнение в главном программном потоке, начиная с Honeycomb и последующих версий Android. Если вы попытаетесь нарушить это ограничение, Android выдает исключение `NetworkOnMainThreadException`. Почему? Чтобы понять это, необходимо понимать, что такое программный поток (thread), что собой представляет главный программный поток приложения и что он делает.

Программным потоком (thread) называется отдельная последовательность выполнения программы. Жизненный цикл каждого приложения Android начинается с главного потока. Однако главный поток не является заранее определенной последовательностью действий. Он в бесконечном цикле ожидает событий, инициированных пользователем или системой, и выполняет код как реакцию на эти события по мере их возникновения.

Представьте, что ваше приложение — огромный обувной магазин, и у вас всего один работник: Флэш[1]. В магазине необходимо многое делать для того, чтобы покупатели

[1] Персонаж комиксов, наделенный способностью к сверхъестественно быстрым перемещениям. — *Примеч. перев.*

остались довольны — расставлять товары на полках, приносить обувь для примерки, определять размер ноги покупателя. С таким продавцом, как Флэш, все делается своевременно, хотя всю работу выполняет всего один человек.

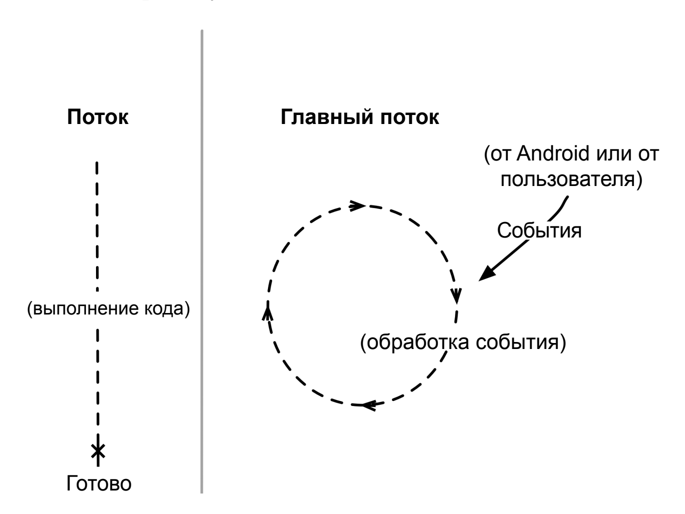


Рис. 26.6. Обычные потоки и главный поток

Чтобы эта ситуация работала, Флэш не может тратить слишком много времени на что-то одно. Что, если часть товара будет распродана? Кому-то придется потратить много времени за беседой по телефону, заказывая у поставщика новую партию. Пока Флэш будет занят, покупатели начнут сердиться.

Флэш — аналог главного потока вашего приложения. Он выполняет весь код, обновляющий пользовательский интерфейс. В частности, сюда относится код реакции на различные события пользовательского интерфейса — запуск активностей, нажатия кнопок и т. д. (Поскольку все события тем или иным образом связаны с пользовательским интерфейсом, главный поток иногда называется потоком пользовательского интерфейса, или UI-потоком.)

Цикл событий обеспечивает последовательное выполнение кода пользовательского интерфейса. Он гарантирует, что операции не будут «перебегать дорогу» друг другу, одновременно обеспечивая своевременное выполнение кода. Таким образом, весь написанный вами код (за исключением кода, написанного для `AsyncTask`) выполнялся в главном потоке.

Кроме главного потока

Сетевые операции можно сравнить с телефонным звонком поставщику обуви: они занимают много времени по сравнению с другими задачами. В это время пользовательский интерфейс будет полностью парализован, что может привести к ошибке ANR (Application Not Responding). Эта ошибка происходит тогда, когда система мониторинга Android обнаруживает, что главный поток не среагировал на важное событие — такое, как нажатие кнопки Back. Для пользователя это выглядит так.

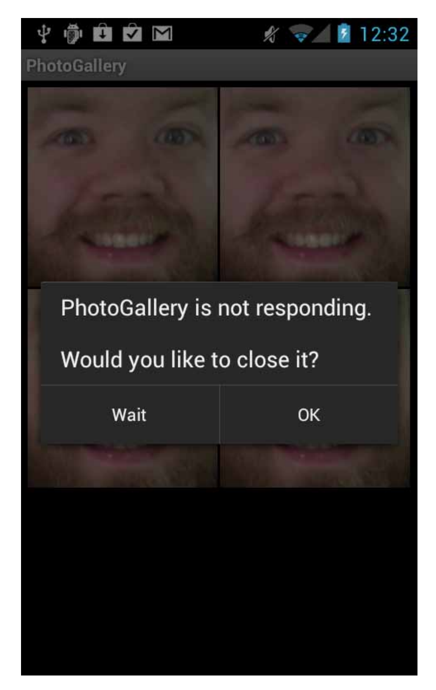


Рис. 26.7. Приложение не отвечает

Вот почему в Android, начиная с версии Honeycomb, запрещено выполнение сетевых операций в главном потоке. В обувном магазине вы бы решили проблему вполне естественным образом — наймом второго работника, который будет звонить поставщику. Похожее решение используется и в Android — вы создаете фоновый поток и выполняете сетевые операции из него.

А как проще всего работать с фоновым потоком? При помощи `AsyncTask`.

Позднее в этой главе вы увидите, на что еще способен класс `AsyncTask`. Но прежде чем изучать его возможности, давайте выполним какую-нибудь реальную работу в сетевом коде.

Загрузка XML из Flickr

Flickr предлагает удобный XML API. Вся необходимая информация доступна в документации по адресу *www.flickr.com/services/api/*. Загрузите ее в своем браузере и найдите список Request Formats. Мы будем использовать простейший формат — REST, соответственно конечной точкой API становится адрес *http://api.flickr.com/services/rest/*. Вы можете вызывать методы, которые Flickr предоставляет в этой конечной точке.

Вернитесь к главной странице документации API и найдите список API Methods. Прокрутите его до раздела photos и найдите элемент flickr.photos.getRecent. Щелкните на нем; в открывшейся документации говорится, что этот метод «Возвращает список

последних общедоступных фотографий, отправленных на flickr». Это именно то, что нам нужно для PhotoGallery.

Единственным обязательным параметром метода `getRecent` является ключ API. Чтобы получить ключ API, обратитесь по адресу *http://www.flickr.com/services/api/* и проследуйте по ссылке API keys. Для входа вам понадобится идентификатор Yahoo. После входа зарегистрируйте новый некоммерческий ключ API; обычно эта процедура занимает несколько секунд. Ваш ключ API будет выглядеть примерно так: 4f721bgafa75bf6d2cb9af54f937bb70.

После получения ключа вам остается лишь обратиться с запросом к веб-службе Flickr. Используйте GET-запрос по адресу *http://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=xxx*.

Переходим к программированию. Начнем с добавления нескольких констант в FlickrFetchr.

Листинг 26.7. Добавление констант (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "yourApiKeyHere";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";

    private static final String EXTRA_SMALL_URL = "url_s";
```

Эти константы определяют конечную точку, имя метода, ключ API и один дополнительный параметр с именем `extras` и значением `url_s`. Задание дополнения `url_s` приказывает Flickr включить URL уменьшенной версии изображения, если она доступна.

Используйте эти константы для написания метода, который строит соответствующий URL-адрес запроса и загружает его содержимое.

Листинг 26.8. Добавление метода fetchItems() (FlickrFetchr.java)

```
public class FlickrFetchr {

    ...

    String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                    .appendQueryParameter("method", METHOD_GET_RECENT)
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                    .build().toString();
            String xmlString = getUrl(url);
```

```
            Log.i(TAG, "Received xml: " + xmlString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

Здесь мы используем класс `Uri.Builder` для построения полного URL-адреса для API-запроса к Flickr. `Uri.Builder` — вспомогательный класс для создания параметризованных URL-адресов с правильным кодированием символов. Метод `Uri.Builder.appendQueryParameter(String,String)` автоматически кодирует строки запросов.

Наконец, измените код `AsyncTask` в `PhotoGalleryFragment` для вызова нового метода `fetchItems()`.

Листинг 26.9. Вызов fetchItems() (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            String result = new FlickrFetchr().getUrl("http://www.google.com");
            Log.i(TAG, "Fetched contents of URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch URL: ", ioe);
        }
        new FlickrFetchr().fetchItems();
        return null;
    }
}
```

Запустите приложение PhotoGallery. В LogCat отображается полноценная, настоящая разметка XML.

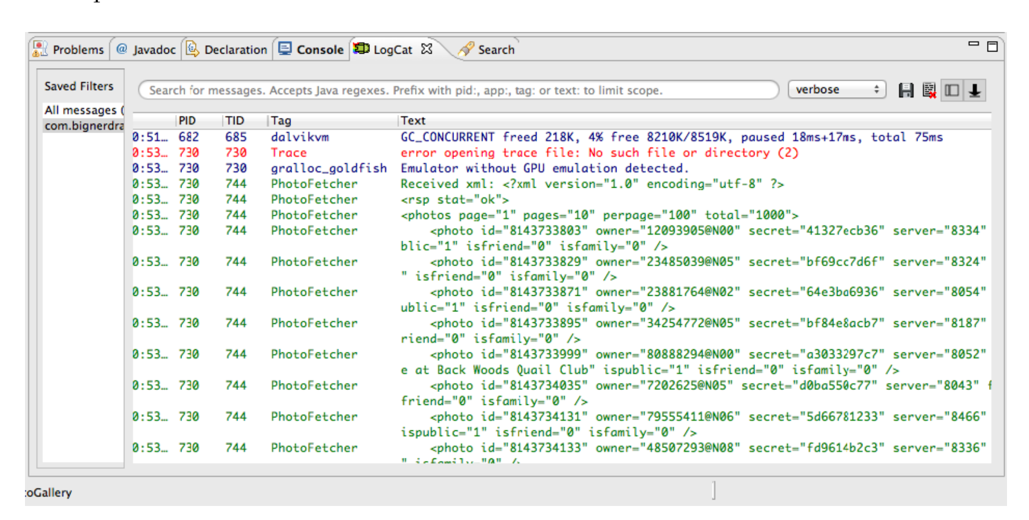


Рис. 26.8. Разметка XML в Flickr

Итак, разметка XML с Flickr получена; что теперь с ней делать? То же, что мы делаем со всеми данными — пометить в один или несколько объектов модели. Класс модели, который мы создадим для PhotoGallery, называется `GalleryItem`. На рис. 26.9 изображена диаграмма объектов PhotoGallery.

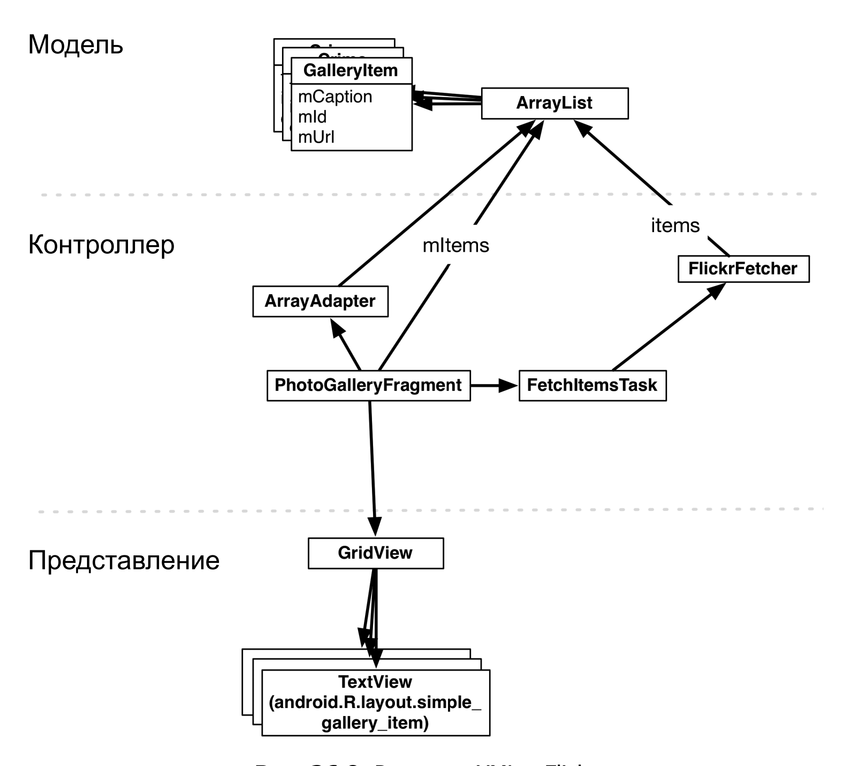


Рис. 26.9. Разметка XML в Flickr

Обратите внимание: на рис. 26.9 не показана активность-хост, чтобы диаграмма была сконцентрирована на фрагментах и сетевом коде.

Создайте класс `GalleryItem` и добавьте следующий код.

Листинг 26.10. Создание класса объекта модели (GalleryItem.java)

```
package com.bignerdranch.android.photogallery;
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    public String toString() {
        return mCaption;
    }
}
```

Прикажите Eclipse сгенерировать `get-` и `set`-методы для `mId`, `mCaption` и `mUrl`.

После того как объекты модели будут созданы, их следует заполнить данными из разметки XML, полученной от Flickr. Для получения данных из XML используется интерфейс `XmlPullParser`.

Использование XmlPullParser

`XmlPullParser` — интерфейс, который может использоваться для выделения событий разбора в потоке разметки XML. `XmlPullParser` используется во внутренней реализации Android для заполнения ваших файлов макетов. С таким же успехом его можно использовать для разбора объектов `GalleryItem`.

Добавьте в `FlickrFetchr` константу, определяющую имя элемента XML, содержащего фотографию. Затем напишите метод, который использует `XmlPullParser` для идентификации всех фотографий в XML. Создайте объект `GalleryItem` для каждой фотографии и добавьте его в `ArrayList`.

Листинг 26.11. Разбор фотографий Flickr (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";
    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "your API key";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";

    private static final String XML_PHOTO = "photo";

    ...

    public void fetchItems() {
        ...
    }

    void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
            throws XmlPullParserException, IOException {
        int eventType = parser.next();

        while (eventType != XmlPullParser.END_DOCUMENT) {
            if (eventType == XmlPullParser.START_TAG &&
                XML_PHOTO.equals(parser.getName())) {
                String id = parser.getAttributeValue(null, "id");
                String caption = parser.getAttributeValue(null, "title");
                String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);

                GalleryItem item = new GalleryItem();
                item.setId(id);
                item.setCaption(caption);
                item.setUrl(smallUrl);
                items.add(item);
            }

            eventType = parser.next();
        }
    }
}
```

Представьте, что `XmlPullParser` водит пальцем по документу XML, последовательно перебирая различные события, такие как `START_TAG`, `END_TAG` и `END_DOCUMENT`. На каждом шаге вы можете вызывать такие методы, как `getText()`, `getName()` или `getAttributeValue(…)`, для получения информации о событии, на которое в настоящий момент указывает `XmlPullParser`. Чтобы переместить `XmlPullParser` к следующему интересному событию в XML, вызовите `next()`. Кстати, этот метод также возвращает тип события, к которому он только что переместился.

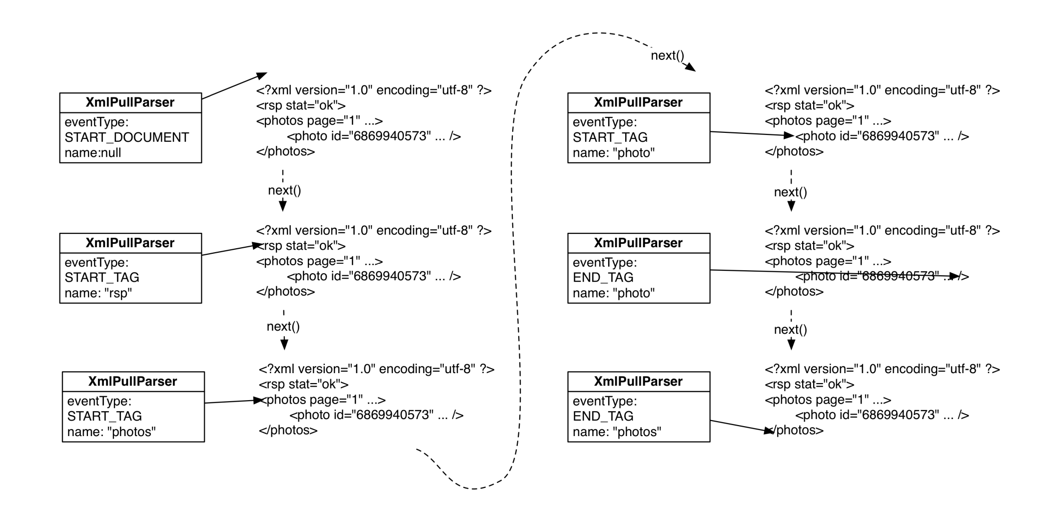


Рис. 26.10. Как работает XmlPullParser

Методу `parseItems(…)` передаются `XmlPullParser` и `ArrayList`. Получите экземпляр парсера и передайте ему данные `xmlString`, полученные от Flickr. Затем вызовите `parseItems(…)` с настроенным парсером и пустым массивом.

Листинг 26.12. Вызов parseItems(...) (FlickrFetchr.java)

```
public void fetchItems() {
public ArrayList<GalleryItem> fetchItems() {
    ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();

    try {
        String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_GET_RECENT)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .build().toString();
        String xmlString = getUrl(url);
        Log.i(TAG, "Received xml: " + xmlString);
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
        XmlPullParser parser = factory.newPullParser();
        parser.setInput(new StringReader(xmlString));

        parseItems(items, parser);
    } catch (IOException ioe) {
```

```
        Log.e(TAG, "Failed to fetch items", ioe);
    } catch (XmlPullParserException xppe) {
        Log.e(TAG, "Failed to parse items", xppe);
    }
    return items;
}
```

Запустите приложение PhotoGallery и протестируйте код разбора XML. У PhotoGallery пока нет средств для вывода информации о содержимом `ArrayList`, поэтому если вы захотите убедиться в том, что все работает правильно, вам придется установить точку прерывания в отладчике.

От AsyncTask к главному потоку

Напоследок мы вернемся к уровню представления и выведем некоторые названия фотографий в виджете `GridView` экземпляра `PhotoGalleryFragment`.

Класс `GridView`, как и `ListView`, происходит от `AdapterView`, поэтому ему нужен адаптер, поставляющий отображаемые представления.

В файле PhotoGalleryFragment.java добавьте объявление `ArrayList` с элементами `GalleryItems`, а затем создайте экземпляр `ArrayAdapter` для простого макета, предоставляемого Android.

Листинг 26.13. Реализация setupAdapter() (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList<GalleryItem> mItems;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        setupAdapter();

        return v;
    }

    void setupAdapter() {
        if (getActivity() == null || mGridView == null) return;

        if (mItems != null) {
            mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
                    android.R.layout.simple_gallery_item, mItems));
        } else {
            mGridView.setAdapter(null);
        }
    }
```

Так как у `GridView` нет удобного класса `GridFragment`, нам придется самостоятельно построить код управления адаптером. Например, для этого можно использовать метод наподобие только что добавленного метода `setupAdapter()`. Этот метод проверяет текущее состояние модели и соответствующим образом настраивает адаптер `GridView`. Этот метод должен вызываться в `onCreateView(…)`, чтобы при каждом создании нового экземпляра `GridView` при повороте он заново настраивался соответствующим адаптером. Метод также должен вызываться при каждом изменении набора объектов модели.

Макет `android.R.layout.simple_gallery_item` состоит из элемента `TextView`. Вспомните, что в `GalleryItem` мы переопределили метод `toString()` для возвращения значения `mCaption` объекта. Таким образом, для отображения названий фотографий в `GridView` достаточно передать адаптеру массив экземпляров `GalleryItem` и этот макет.

Обратите внимание: перед назначением адаптера мы проверяем `getActivity()` на `null`. Помните, что фрагменты могут существовать и автономно, не будучи связанными с какой-либо активностью. До настоящего момента мы не сталкивались с такой возможностью, потому что вызовы методов управлялись обратными вызовами от инфраструктуры. Если фрагмент получает обратные вызовы, он определенно присоединен к активности. Нет активности — нет обратных вызовов.

Теперь, когда мы используем `AsyncTask`, некоторые действия инициируются самостоятельно, и мы уже не можем предполагать, что фрагмент присоединен к активности. Таким образом, сначала необходимо убедиться в том, что фрагмент остается присоединенным; в противном случае попытка выполнения операции завершится неудачей.

После получения данных от Flickr следует вызвать метод `setupAdapter()`. Первое, что приходит в голову, — вызов `setupAdapter()` в конце метода `doInBackground(…)` класса `FetchItemsTask`. Это не лучшая мысль. Вспомните, что сейчас «в магазине работают два Флэша» — один обслуживает многочисленных покупателей, другой общается по телефону с Flickr. Что произойдет, если второй Флэш, повесив трубку, захочет подключиться к обслуживанию покупателей? Скорее всего, два Флэша только начнут мешать друг другу.

На компьютере такая путаница может привести к повреждению объектов в памяти. По этой причине фоновым потокам запрещается обновлять пользовательский интерфейс, поскольку такие операции явно небезопасны.

Что делать? У `AsyncTask` имеется метод `onPostExecute(…)`, который можно переопределить. Метод `onPostExecute(…)` выполняется после завершения `doInBackground(…)`. Кроме того, `onPostExecute(…)` выполняется в главном, а не в фоновом потоке, поэтому обновление пользовательского интерфейса в нем безопасно.

Внесите изменения в метод `FetchItemsTask`, чтобы он обновлял поле `mItems` и вызывал `setupAdapter()` после загрузки фотографий.

Листинг 26.14. Добавление кода обновления адаптера (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    protected Void doInBackground(Void... params) {
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
```

```
            new FlickrFetchr().fetchItems();
            return new FlickrFetchr().fetchItems();
            return null;
        }

        @Override
        protected void onPostExecute(ArrayList<GalleryItem> items) {
            mItems = items;
            setupAdapter();
        }
    }
```

Мы внесли три изменения. Во-первых, мы изменили тип третьего обобщенного параметра `FetchItemsTask`. Этот параметр определяет тип результата, производимого `AsyncTask`; он задает тип значения, возвращаемого `doInBackground(…)`, а также тип входного параметра `onPostExecute(…)`. Во-вторых, мы изменили метод `doInBackground(…)` так, чтобы он возвращал список элементов `GalleryItem`. Тем самым мы устранили ошибку в коде и обеспечили его нормальную компиляцию. Также при вызове передается список элементов, чтобы он мог использоваться в коде `onPostExecute(…)`. В-третьих, была добавлена реализация `onPostExecute(…)`. Этот метод получает список, загруженный в `doInBackground(…)`, помещает его в `mItems` и обновляет адаптер `GridView`.

На этом наша работа для этой главы завершается. Запустите приложение, и вы увидите текст, отображаемый для каждого загруженного элемента `GalleryItem`.



Рис. 26.11. Заголовки фотографий Flickr

Для любознательных: подробнее об AsyncTask

В этой главе вы видели пример использования последнего параметра-типа `AsyncTask`, определяющего возвращаемый тип. А как насчет двух других параметров?

Первый параметр-тип позволяет задать тип входных параметров. Он используется следующим образом:

```
AsyncTask<String,Void,Void> task = new AsyncTask<String,Void,Void>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }

        return null;
    }
};

task.execute("First parameter", "Second parameter", "Etc.");
```

Входные параметры передаются методу `execute(…)`, который вызывается с переменным числом аргументов. Эти переменные аргументы передаются `doInBackground(…)`.

Второй параметр-тип позволяет задать тип для передачи информации о ходе выполнения операции. Вот как это выглядит:

```
final ProgressBar progressBar = /* Индикатор прогресса */;
progressBar.setMax(100);
AsyncTask<Integer,Integer,Void> task = new AsyncTask<Integer,Integer,Void>() {
    public Void doInBackground(Integer... params) {
        for (Integer progress : params) {
            publishProgress(progress);
            Thread.sleep(1000);
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        progressBar.setProgress(progress);
    }
};
task.execute(25, 50, 75, 100);
```

Обновление обычно выполняется в продолжительном фоновом процессе. Проблема в том, что необходимые обновления пользовательского интерфейса не могут выполняться прямо из фонового процесса, поэтому `AsyncTask` предоставляет методы `publishProgress(…)` и `onProgressUpdate(…)`.

Механизм обновления работает следующим образом: в методе `doInBackground(…)` в фоновом потоке вызывается `publishProgress(…)`. Это приводит к вызову `onProgressUpdate(…)` в потоке пользовательского интерфейса. Таким образом, пользовательский интерфейс обновляется в `onProgressUpdate(…)`, но управление обновлениями осуществляется вызовами `publishProgress(…)` в `doInBackground(…)`.

Уничтожение AsyncTask

В этой главе реализация AsyncTask была тщательно структурирована таким образом, чтобы нам не приходилось хранить информацию о ней. Однако в других ситуациях может возникнуть необходимость в отслеживании AsyncTask и даже их периодической отмене и повторном запуске.

В подобных более сложных сценариях использования AsyncTask присваивается переменной экземпляра, после чего для нее можно вызвать AsyncTask.cancel(boolean) для отмены текущей фоновой операции AsyncTask.

AsyncTask.cancel(boolean) может работать в более жестком или менее жестком режиме. Если вызвать cancel(false), метод действует менее жестко и просто возвращает true при вызове isCancelled(). Далее AsyncTask проверяет isCancelled() внутри doInBackground(…) и принимает решение о досрочном завершении операции.

Но в случае вызова cancel(true) метод действует жестко и прерывает программный поток, в котором выполняется doInBackground(…). Вызов AsyncTask.cancel(true) является агрессивным способом остановки AsyncTask. Если этого можно избежать — лучше так и сделать.

Упражнение. Страничная навигация

По умолчанию метод getRecent возвращает одну страницу со 100 результатами. При помощи дополнительного параметра page получить вторую, третью и так далее страницу результатов.

Добавьте в адаптер код, который обнаруживает переход к концу массива элементов и заменяет текущую страницу следующей страницей результатов. Чтобы немного усложнить упражнение, организуйте присоединение данных последующих страниц к результатам.

27 Looper, Handler и HandlerThread

После загрузки и разбора разметки XML от Flickr нашей следующей задачей станет загрузка и вывод изображений. В этой главе вы научитесь использовать классы `Looper`, `Handler` и `HandlerThread` для динамической загрузки и вывода фотографий в PhotoGallery.

Подготовка GridView к выводу изображений

Текущий адаптер `PhotoGalleryFragment` просто предоставляет виджеты `TextView` для вывода в `GridView`. В каждом представлении `TextView` выводится содержимое заголовка `GalleryItem`.

Чтобы выводить фотографии, вам понадобится нестандартный адаптер, который вместо текстовых представлений предоставляет `ImageView`. В конечном итоге `ImageView` выведет фотографию, загруженную в поле `mUrl` экземпляра `GalleryItem`.

Начнем с создания нового файла макета для элементов фотогалереи в файле gallery_item.xml. Макет будет состоять из единственного виджета `ImageView` (рис. 27.1).

ImageView
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:id="**@+id/gallery_item_imageView**"
android:layout_width="**match_parent**"
android:layout_height="**120dp**"
android:layout_gravity="**center**"
android:scaleType="**centerCrop**"

Рис. 27.1. Макет элемента фотогалереи (res/layout/gallery_item.xml)

Эти виджеты ImageView находятся под управлением GridView, это означает, что их ширина будет величиной переменной. Тем не менее высота будет оставаться фиксированной. Чтобы наиболее эффективно использовать пространство виджета ImageView, следует задать его свойству scaleType значение centerCrop. С этим значением изображение выравнивается по центру и масштабируется, чтобы меньшая сторона была равна размеру представления, а большая усекалась с обеих сторон.

Также нам понадобится временное изображение для каждого виджета ImageView, которое будет отображаться до завершения загрузки изображения. Найдите файл brian_up_close.jpg в файле решений и поместите его в каталог res/drawable-hdpi.

В классе PhotoGalleryFragment замените базовую реализацию ArrayAdapter пользовательской реализацией, у которой реализация getView(…) возвращает виджет ImageView с временным изображением.

Листинг 27.1. Создание GalleryItemAdapter (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...

    void setupAdapter() {
        if (getActivity() == null || mGridView == null) return;

        if (mItems != null) {
            mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
                    android.R.layout.simple_gallery_item, mItems));
            mGridView.setAdapter(new GalleryItemAdapter(mItems));
        } else {
            mGridView.setAdapter(null);
        }
    }

    private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
        ...
    }

    private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
        public GalleryItemAdapter(ArrayList<GalleryItem> items) {
            super(getActivity(), 0, items);
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            if (convertView == null) {
                convertView = getActivity().getLayoutInflater()
                        .inflate(R.layout.gallery_item, parent, false);
            }

            ImageView imageView = (ImageView)convertView
                    .findViewById(R.id.gallery_item_imageView);
            imageView.setImageResource(R.drawable.brian_up_close);

            return convertView;
        }
    }
}
```

Не забудьте, что AdapterView (GridView в данном случае) вызывает метод getView(...) своего адаптера для каждого нужного ему отдельного представления.

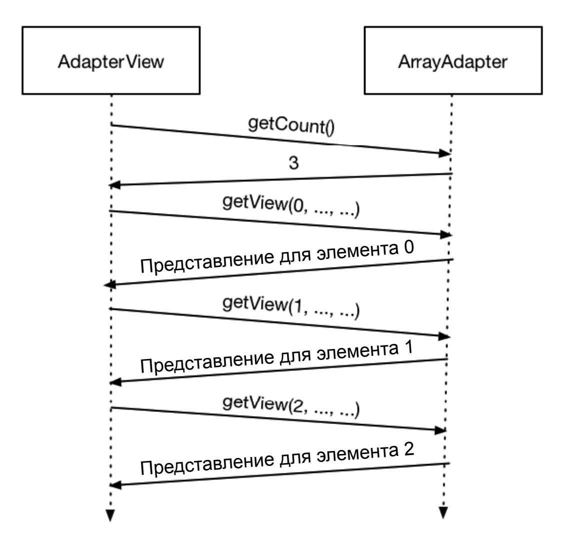


Рис. 27.2. Взаимодействия AdapterView-ArrayAdapter

Запустив приложение PhotoGallery, вы увидите набор увеличенных Брайанов.



Рис. 27.3. Брайаны повсюду

Множественные загрузки

В настоящее время сетевая часть PhotoGallery работает следующим образом: `PhotoGalleryFragment` запускает экземпляр `AsyncTask`, который получает XML от Flickr в фоновом потоке, и разбирает XML в массив объектов `GalleryItem`. В каждом объекте `GalleryItem` теперь хранится URL, по которому находится миниатюрная версия фотографии.

Следующим шагом должна стать загрузка этих миниатюр. Казалось бы, дополнительный сетевой код можно просто добавить в метод `doInBackground()` класса `FetchItemsTask`. Массив объектов `GalleryItem` содержит 100 URL-адресов для загрузки. Изображения будут загружаться одно за одним, пока у вас не появятся все 100. При выполнении `onPostExecute(…)` они все вместе появятся в `GridView`.

Однако единовременная загрузка всех миниатюр создает две проблемы. Во-первых, она займет довольно много времени, а пользовательский интерфейс не будет обновляться до момента ее завершения. На медленном подключении пользователям придется долго рассматривать стену Брайанов.

Во-вторых, хранение полного набора изображений требует ресурсов. Сотня миниатюр легко уместится в памяти. Но что, если их будет 1000? Что, если вы захотите реализовать бесконечную прокрутку? Со временем свободная память будет исчерпана.

С учетом этих проблем реальные приложения часто загружают изображения только тогда, когда они должны выводиться на экране. Загрузка по мере надобности предъявляет дополнительные требования к `GridView` и его адаптеру. Загрузка изображения инициируется как часть реализации `getView(…)` адаптера.

`AsyncTask` — самый простой способ получения фоновых потоков, но для многократно выполняемых и продолжительных операций этот механизм изначально малопригоден. (О том, почему это так, рассказано в разделе «Для любознательных» в конце этой главы.)

Вместо использования `AsyncTask` мы создадим специализированный фоновый поток. Это самый распространенный способ реализации загрузки по мере надобности.

Взаимодействие с главным потоком

Специализированный поток будет загружать фотографии, но как он будет взаимодействовать с адаптером `GridView` для их отображения, если он не может напрямую обращаться к главному потоку?

Вспомните пример с обувным магазином и двумя продавцами-Флэшами. Фоновый Флэш завершил свой телефонный разговор с поставщиков и теперь ему нужно сообщить Главному Флэшу о том, что обувь была заказана. Если Главный Флэш занят, Фоновый Флэш не может сделать это немедленно. Ему придется подождать у стойки и перехватить Главного Флэша в свободный момент. Такая схема работает, но не слишком эффективно.

Лучше дать каждому Флэшу по почтовому ящику. Фоновый Флэш пишет сообщение о том, что обувь заказана, и кладет его в ящик Главного Флэша. Главный Флэш делает то же самое, когда он хочет сообщить Фоновому Флэшу о том, что какой-то товар закончился.

Идея почтового ящика чрезвычайно полезна. Возможно, у продавца имеется задача, которая должна быть выполнена скоро, но не прямо сейчас. В таком случае он кладет сообщение в свой почтовый ящик и обрабатывает его в свободное время.

В Android такой «почтовый ящик», используемый потоками, называется *очередью сообщений* (message queue). Поток, работающий с использованием очереди сообщений, называется *циклом сообщений* (message loop); он снова и снова проверяет новые сообщения, которые могли появиться в очереди (рис. 27.4).

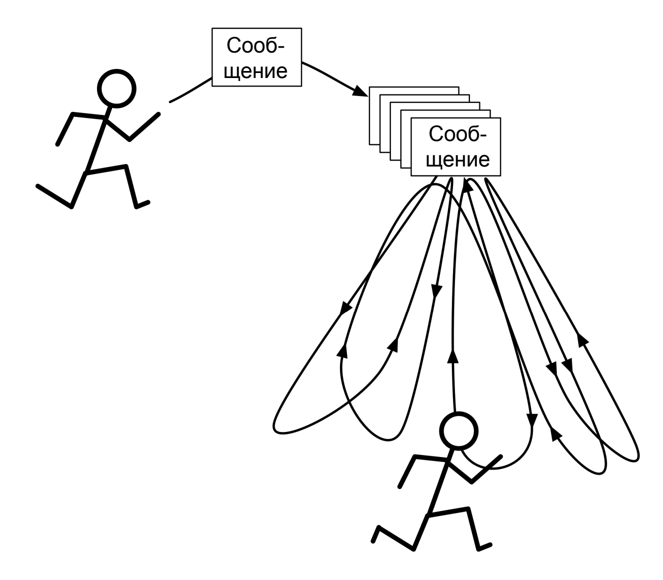


Рис. 27.4. Цикл сообщений

Цикл сообщений состоит из потока и объекта `Looper`, управляющего очередью сообщений потока.

Главный поток представляет собой цикл сообщений и у него есть управляющий объект, который извлекает сообщения из очереди сообщений и выполняет задачу, описанную в сообщении.

Мы создадим фоновый поток, который тоже использует цикл сообщений. При этом будет использоваться класс `HandlerThread`, который предоставляет готовый объект `Looper`.

Создание фонового потока

Создайте новый класс с именем `ThumbnailDownloader`, расширяющий `HandlerThread`. Пользователю `ThumbnailDownloader` понадобится объект для идентификации

каждой загрузки; определите один обобщенный аргумент Token, присвоив ему имя ThumbnailDownloader<Token> в диалоговом окне создания класса. Затем определите для него конструктор и заглушку реализации метода с именем queueThumbnail() (листинг 27.2).

Листинг 27.2. Исходная версия кода потока (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    public ThumbnailDownloader() {
        super(TAG);
    }

    public void queueThumbnail(Token token, String url) {
        Log.i(TAG, "Got an URL: " + url);
    }
}
```

Обратите внимание: queueThumbnail() ожидает получить Token и String. Этот метод будет вызываться GalleryItemAdapter в его реализации getView(…).

Откройте файл PhotoGalleryFragment.java. Определите в PhotoGalleryFragment поле типа ThumbnailDownloader. В качестве маркера (token) ThumbnailDownloader можно использовать любой объект. В нашем случае удобным маркером является объект ImageView, так как в конечном итоге загруженные изображения попадут именно в него. В методе onCreate(…) создайте поток и запустите его. Переопределите метод onDestroy() для завершения потока.

Листинг 27.3. Создание класса ThumbnailDownloader (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList<GalleryItem> mItems;
    ThumbnailDownloader<ImageView> mThumbnailThread;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
        new FetchItemsTask().execute();

        mThumbnailThread = new ThumbnailDownloader<ImageView>();
        mThumbnailThread.start();
        mThumbnailThread.getLooper();
        Log.i(TAG, "Background thread started");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }
```

продолжение ⇨

Листинг 27.3 (*продолжение*)

```
    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailThread.quit();
        Log.i(TAG, "Background thread destroyed");
    }
    ...
}
```

Пара примечаний: во-первых, обратите внимание на то, что вызов `getLooper()` следует после вызова `start()` для `ThumbnailDownloader`. Тем самым гарантируется, что внутреннее состояние потока готово для продолжения (вскоре мы рассмотрим объект `Looper` более подробно). Во-вторых, вызов `quit()` завершает поток внутри `onDestroy()`. Это очень важный момент. Если не завершать потоки `HandlerThread`, они никогда не умрут, словно зомби).

Наконец, в методе `GalleryItemAdapter.getView(…)` получите правильный элемент `GalleryItem` с использованием параметра `position`, вызовите метод `queueThumbnail()` потока и передайте `ImageView` и URL-адрес элемента.

Листинг 27.4. Подключение ThumbnailDownloader (PhotoGalleryFragment.java)

```
private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
    ...

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ...

        ImageView imageView = (ImageView)convertView
                .findViewById(R.id.gallery_item_imageView);
        imageView.setImageResource(R.drawable.brian_up_close);
        GalleryItem item = getItem(position);
        mThumbnailThread.queueThumbnail(imageView, item.getUrl());

        return convertView;
    }
}
```

Запустите приложение PhotoGallery и проверьте данные LogCat. При прокрутке `GridView` в LogCat появляются строки, сообщающие о том, что `ThumbnailDownloader` получает все запросы на загрузку.

Теперь, когда наша реализация `HandlerThread` заработала, следующим шагом становится создание сообщения с информацией, переданной `queueThumbnail()`, и его размещение в очереди сообщений `ThumbnailDownloader`.

Сообщения и обработчики сообщений

Прежде чем создавать сообщение, необходимо сначала понять, что оно собой представляет и какие отношения связывают его с *обработчиком сообщения* (message handler).

Строение сообщения

Начнем с сообщений. Сообщения, которые Флэш кладет в почтовый ящик (свой собственный или принадлежащий другому продавцу), содержат не ободряющие записки типа «Ты бегаешь очень быстро», а описания задач, которые необходимо выполнить.

Сообщение является экземпляром класса `Message` и состоит из нескольких полей. Для нашей реализации важны три поля:

- `what` — определяемое пользователем значение `int`, описывающее сообщение;
- `obj` — заданный пользователем объект, передаваемый с сообщением;
- `target` — приемник, то есть объект `Handler`, который будет обрабатывать сообщение.

Приемником сообщения `Message` является экземпляр `Handler`. Когда вы создаете сообщение, оно автоматически присоединяется к `Handler`. А когда ваше сообщение будет готово к обработке, именно `Handler` становится объектом, отвечающим за эту обработку.

Строение обработчика

Итак, для выполнения реальной работы с сообщениями необходимо иметь экземпляр `Handler`. Объект `Handler` — не просто приемник для обработки сообщений; он также предоставляет интерфейс для создания и отправки сообщений.

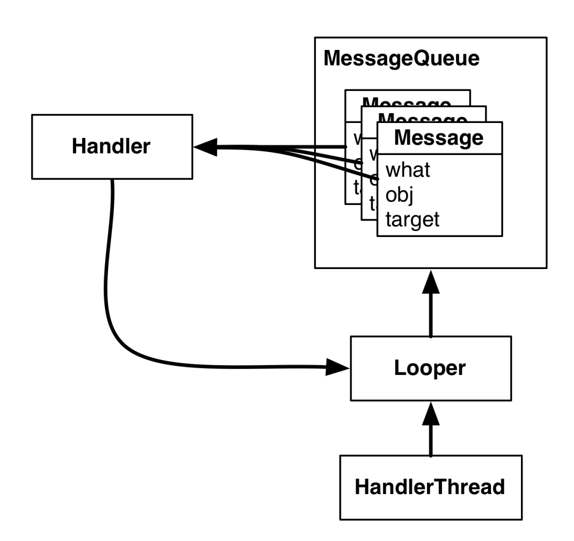


Рис. 27.5. Looper, Handler, HandlerThread и Message

Сообщения `Message` отправляются и потребляются объектом `Looper`, потому что `Looper` является владельцем почтового ящика объектов `Message`. Соответственно `Handler` всегда содержит ссылку на своего коллегу `Looper`.

Handler всегда присоединяется ровно к одному объекту Looper, а Message присоединяется ровно к одному объекту Handler, называемому его *приемником*. Объект Looper обслуживает целую очередь сообщений Message.

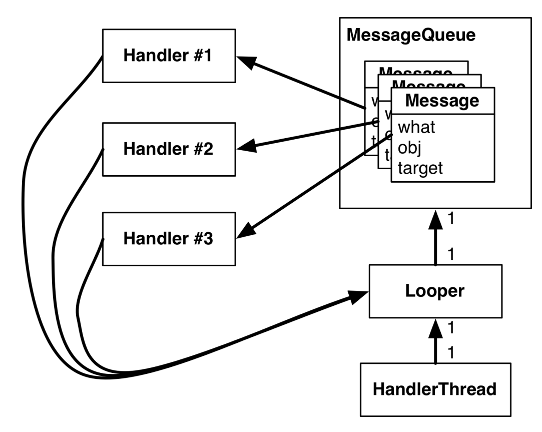


Рис. 27.6. Несколько объектов Handler, один объект Looper

К одному объекту Looper могут быть присоединены несколько объектов Handler. Это означает, что сообщения Message объекта Handler могут сосуществовать с сообщениями другого объекта Handler.

Использование обработчиков

Обычно приемные объекты Handler сообщений не задаются вручную. Лучше построить сообщение вызовом Handler.obtainMessage(…). Вы передаете методу другие поля сообщения, а он автоматически назначает приемник.

Метод Handler.obtainMessage(…) использует общий пул объектов, чтобы избежать создания новых объектов Message, поэтому он также работает более эффективно, чем простое создание новых экземпляров.

Когда объект Message будет получен, вы можете вызвать метод sendToTarget(), чтобы отправить сообщение его обработчику. Обработчик помещает сообщение в конец очереди сообщений объекта Looper.

Мы собираемся получить сообщение и отправить его приемнику в реализации queueThumbnail(). В поле what сообщения будет содержаться константа, определяемая под именем MESSAGE_DOWNLOAD. В поле obj будет содержаться маркер Token — в нашем случае экземпляр ImageView, переданный адаптером queueThumbnail().

Когда объект Looper добирается до конкретного сообщения в очереди, он передает сообщение приемнику сообщения для обработки. Как правило, сообщение обрабатывается в реализации Handler.handleMessage(…) приемника.

В нашем случае реализация handleMessage(…) будет использовать FlickrFetchr для загрузки байтов по URL-адресу и их преобразования в растровое изображение. Добавьте код, приведенный в листинге 27.5.

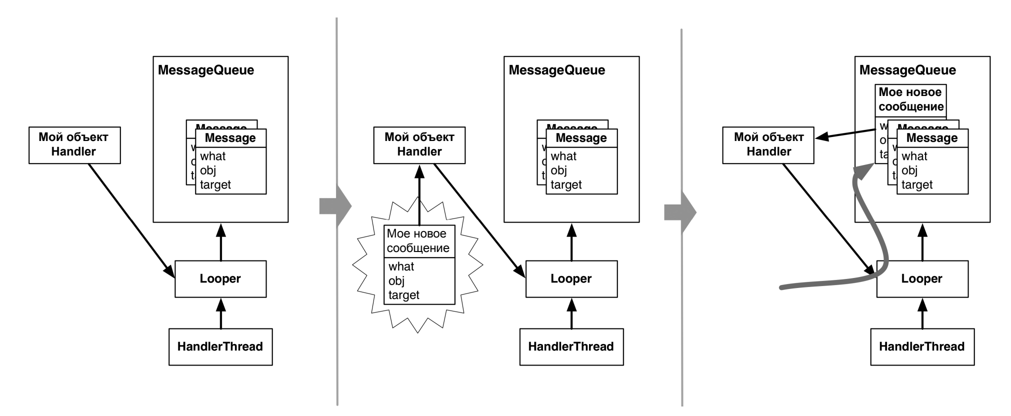


Рис. 27.7. Создание сообщения и его отправка

Листинг 27.5. Получение, отправка и обработка сообщения (ThumbnailDownloader.java)

```
public class ThumbnailDownloader<Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map<Token, String> requestMap =
            Collections.synchronizedMap(new HashMap<Token, String>());

    public ThumbnailDownloader() {
        super(TAG);
    }

    @SuppressLint("HandlerLeak")
    @Override
    protected void onLooperPrepared() {
        mHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                if (msg.what == MESSAGE_DOWNLOAD) {
                    @SuppressWarnings("unchecked")
                    Token token = (Token)msg.obj;
                    Log.i(TAG, "Got a request for url: " + requestMap.get(token));
                    handleRequest(token);
                }
            }
        };
    }

    public void queueThumbnail(Token token, String url) {
        Log.i(TAG, "Got a URL: " + url");
        requestMap.put(token, url);

        mHandler
            .obtainMessage(MESSAGE_DOWNLOAD, token)
            .sendToTarget();
    }
```

продолжение ⇨

Листинг 27.5 (*продолжение*)

```
    private void handleRequest(final Token token) {
        try {
            final String url = requestMap.get(token);
            if (url == null)
                return;

            byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
            final Bitmap bitmap = BitmapFactory
                    .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
            Log.i(TAG, "Bitmap created");

        } catch (IOException ioe) {
            Log.e(TAG, "Error downloading image", ioe);
    }
}
```

Для начала обратите внимание на аннотацию в начале `onLooperPrepared()` — `@SuppressLint("HandlerLeak")`. Здесь Android Lint выдает предупреждение о субклассировании `Handler`. Жизнеспособность нашего объекта `Handler` будет обеспечивать его экземпляр `Looper`. Таким образом, если `Handler` представляет собой анонимный внутренний класс, может легко возникнуть утечка памяти через неявную ссылку на объект. Однако в нашей ситуации все привязано к экземпляру `HandlerThread`, так что утечка памяти исключена.

Другая аннотация `@SuppressWarnings("unchecked")` более типична. Она необходима, потому что `Token` представляет собой обобщенный аргумент класса, а `Message.obj` относится к типу `Object`. Из-за стирания типа прямое преобразование такого рода невозможно. Если вы захотите изучить эту тему более подробно, поищите информацию о стирании типов (type erasure) — в этой главе нас интересует программирование для Android.

Переменная `requestMap` представляет собой синхронизированный объект `HashMap`. Здесь, используя `Token` как ключ, можно хранить и загружать URL-адреса, связанные с конкретным объектом `Token`.

В методе `queueThumbnail()` в хеш-карту добавляется переданная пара «`Token`-URL». Затем мы получаем сообщение, передаем `Token` как значение `obj` и отправляем на постановку в очередь сообщений.

Мы реализуем `Handler.handleMessage(…)` в субклассе `Handler` внутри `onLooperPrepared()`. Метод `HandlerThread.onLooperPrepared()` вызывается до того, как `Looper` проверит очередь в первый раз, поэтому он хорошо подходит для создания реализации `Handler`.

В `Handler.handleMessage(…)` мы проверяем тип сообщения, получаем маркер `Token` и передаем его `handleRequest(…)`.

Вся загрузка осуществляется в методе `handleRequest()`. Мы проверяем существование URL-адреса, после чего передаем его новому экземпляру знакомого класса `FlickrFetchr`. При этом используется метод `FlickrFetchr.getUrlBytes(…)`, который мы так предусмотрительно создали в последней главе.

Наконец, мы используем класс BitmapFactory для построения растрового изображения с массивом байтов, возвращенным getUrlBytes(…).

Запустите приложение PhotoGallery и проверьте в данных LogCat ваши подтверждающие команды регистрации.

Разумеется, запрос не будет полностью обработан до момента назначения изображения в виджете ImageView, поступившем от GalleryItemAdapter. Однако эта операция относится к пользовательскому интерфейсу, поэтому она должна выполняться в главном потоке.

До настоящего момента мы ограничивались использованием обработчиков и сообщений в одном потоке — помещением сообщений в собственный почтовый ящик. В следующем разделе вы увидите, как ThumbnailDownloader использует Handler для обращения к главному потоку.

Передача Handler

Один из способов, которым HandlerThread может выполнить работу в главном потоке, основан на передаче главным потоком своего объекта Handler в HandlerThread.

Главный поток представляет собой цикл сообщений с обработчиками и Looper. При создании экземпляра Handler в главном потоке он ассоциируется с экземпляром Looper главного потока. Затем этот экземпляр Handler можно передать другому потоку. Переданный экземпляр Handler сохраняет связь с Looper потока-создателя.

Все сообщения, за которые отвечает Handler, будут обрабатываться в очереди главного потока. Происходящее очень похоже на то, как мы планировали операции в фоновом потоке из главного потока с использованием Handler экземпляра ThumbnailDownloader.

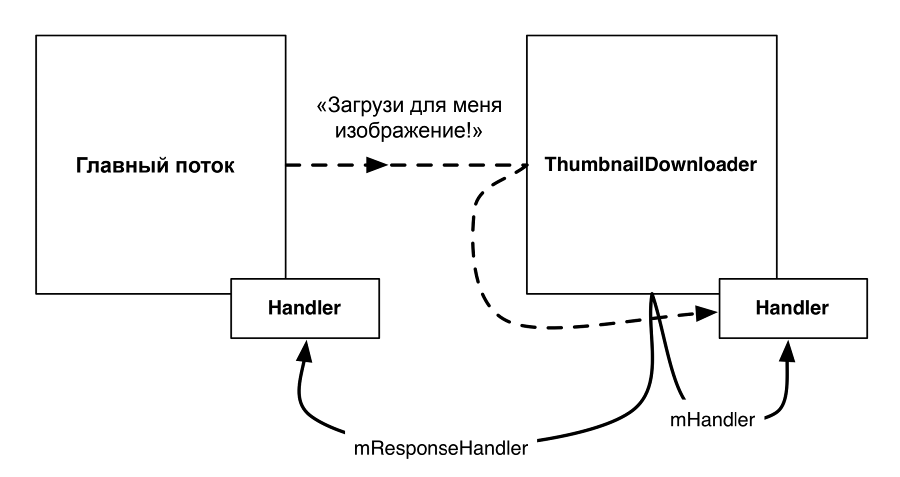


Рис. 27.8. Планирование операций в ThumbnailDownloader из главного потока

Аналогичным образом можно планировать операции в главном потоке из фонового потока с использованием экземпляра Handler, присоединенного к главному потоку.

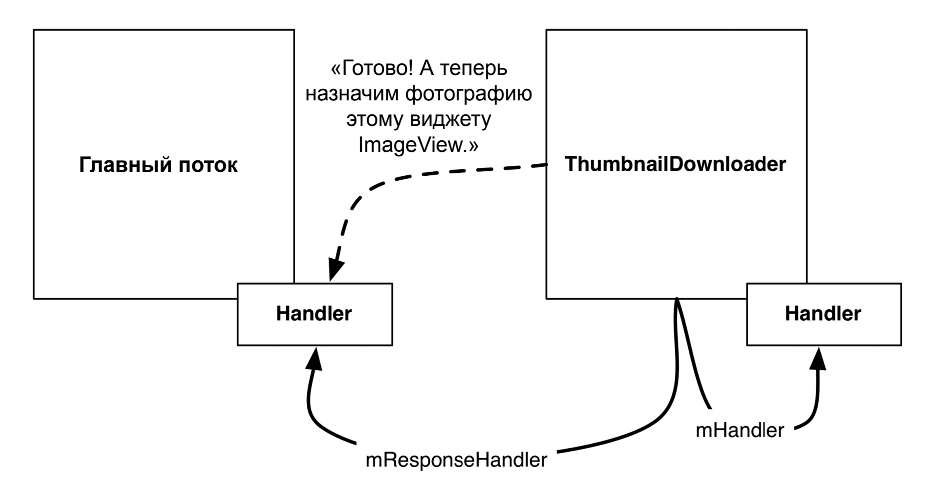


Рис. 27.9. Планирование операций в главном потоке из ThumbnailDownloader

В файле ThumbnailDownloader.java добавьте упоминавшуюся выше переменную mResponseHandler для хранения экземпляра Handler, переданного из главного потока. Затем замените конструктор другим, который получает Handler и задает переменную, и добавьте интерфейс слушателя для передачи ответов.

Листинг 27.6. Добавление обработчика ответа (ThumbnailDownloader.java)

```
public class ThumbnailDownloader extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map<Token,String> requestMap =
            Collections.synchronizedMap(new HashMap<Token,String>());
    Handler mResponseHandler;
    Listener<Token> mListener;

    public interface Listener<Token> {
        void onThumbnailDownloaded(Token token, Bitmap thumbnail);
    }

    public void setListener(Listener<Token> listener) {
        mListener = listener;
    }

    public ThumbnailDownloader() {
        super(TAG);
    public ThumbnailDownloader(Handler responseHandler) {
        super(TAG);
        mResponseHandler = responseHandler;
    }
```

Затем измените класс PhotoGalleryFragment так, чтобы он передавал Handler классу ThumbnailDownloader, а также Listener для задания возвращаемых экземпляров

Bitmap по дескрипторам ImageView. Помните, что по умолчанию Handler присоединяет себя к экземпляру Looper текущего потока. Так как экземпляр Handler создается в onCreate(…), он будет присоединен к экземпляру Looper главного потока.

Листинг 27.7. Подключение к обработчику ответа (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    new FetchItemsTask().execute();

    mThumbnailThread = new ThumbnailDownloader();
    mThumbnailThread = new ThumbnailDownloader(new Handler());
    mThumbnailThread.setListener(new ThumbnailDownloader.Listener<ImageView>() {
        public void onThumbnailDownloaded(ImageView imageView, Bitmap thumbnail) {
            if (isVisible()) {
                imageView.setImageBitmap(thumbnail);
            }
        }
    });
    mThumbnailThread.start();
    mThumbnailThread.getLooper();
    Log.i(TAG, "Background thread started");
}
```

Теперь ThumbnailDownloader имеет доступ к экземпляру Handler, связанному с экземпляром Looper главного потока, через поле mResponseHandler. Кроме того, он приказывает Listener выполнять операции пользовательского интерфейса с возвращаемыми объектами Bitmap. Обратите внимание: вызов imageView.setImageBitmap(Bitmap) защищается вызовом Fragment.isVisible(). Защита предотвращает назначение изображения устаревшему виджету ImageView.

Аналогичным образом можно отправить главному потоку нестандартный объект Messsage. Для этого потребуется другой субкласс Handler с переопределением handleMessage(…). Однако вместо этого мы используем другой удобный метод Handler — post(Runnable).

Handler.post(Runnable) — вспомогательный метод для отправки сообщений следующего вида:

```
Runnable myRunnable = new Runnable() {
    public void run() {
        /* Здесь располагается ваш код */
    }
};
Message m = mHandler.obtainMessage();
m.callback = myRunnable;
```

Если у Message задано поле callback, то вместо приемника Handler выполняется объект Runnable из поля callback.

Включите в ThumbnailDownloader.handleRequest() следующий код.

Листинг 27.8. Загрузка и вывод (ThumbnailDownloader.java)

```
...
private void handleRequest(final Token token) {
    try {
        final String url = requestMap.get(token);
        if (url == null)
            return;

        byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
        final Bitmap bitmap = BitmapFactory
                .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
        Log.i(TAG, "Bitmap created");

        mResponseHandler.post(new Runnable() {
            public void run() {
                if (requestMap.get(token) != url)
                    return;

                requestMap.remove(token);
                mListener.onThumbnailDownloaded(token, bitmap);
            }
        });
    } catch (IOException ioe) {
        Log.e(TAG, "Error downloading image", ioe);
    }
}
```

А поскольку mResponseHandler связывается с Looper главного потока, этот код обновления пользовательского интерфейса будет выполнен в главном потоке.

Что делает этот код? Сначала он проверяет requestMap. Такая проверка необходима, потому что GridView заново использует свои представления. К тому времени, когда ThumbnailDownloader завершит загрузку Bitmap, может оказаться, что виджет GridView уже переработал ImageView и запросил для него изображение с другого URL-адреса. Эта проверка гарантирует, что каждый объект Token получит правильное изображение, даже если за прошедшее время был сделан другой запрос.

Наконец, мы удаляем Token из requestMap и назначаем изображение для Token.

Прежде чем запускать приложение, чтобы увидеть завоеванные тяжелым трудом изображения, необходимо принять во внимание одну последнюю опасность. Если пользователь повернет экран, ThumbnailDownloader может оказаться связанным с недействительными экземплярами ImageView. Нажатие на них грозит всевозможными неприятностями.

Напишите следующий метод для удаления всех запросов из очереди.

Листинг 27.9. Добавление метода очистки очереди (ThumbnailDownloader.java)

```
public void clearQueue() {
    mHandler.removeMessages(MESSAGE_DOWNLOAD);
    requestMap.clear();
}
```

Затем очистите загрузчик в PhotoGalleryFragment при уничтожении представления.

Листинг 27.10. Вызов метода очистки очереди (PhotoGalleryFragment.java)

```
@Override
public void onDestroyView() {
    super.onDestroyView();
    mThumbnailThread.clearQueue();
}
```

На этом наша работа в этой главе подходит к концу. Запустите приложение PhotoGallery. Прокрутите список и понаблюдайте за тем, как происходит динамическая загрузка изображений.

Приложение PhotoGallery выполняет свою основную функцию — вывод изображений из Flickr. В нескольких следующих главах мы дополним его новыми функциями: поиском фотографий и открытием страницы Flickr каждой фотографии в веб-представлении.

Для любознательных: AsyncTask и потоки

Теперь вы понимаете, как работают классы `Handler` и `Looper`, и класс `AsyncTask` уже кажется не таким волшебным. При этом он требует меньшего объема работы по сравнению с тем, что мы сделали сейчас. Так почему бы не использовать `AsyncTask` вместо `HandlerThread`?

По нескольким причинам. Самая принципиальная заключается в том, что класс `AsyncTask` проектировался не для этого. Он предназначен для кратковременной работы, которая не повторяется слишком часто. `AsyncTask` отлично подходит для таких ситуаций, как в нашем коде из предыдущей главы. Но если вы создаете множество `AsyncTask` или они выполняются в течение долгого времени, вероятно, вы неправильно выбрали класс.

Есть и другая, более убедительная техническая причина: в Android 3.2 реализация `AsyncTask` была серьезно изменена. Начиная с Android 3.2, `AsyncTask` не создает поток для каждого экземпляра `AsyncTask`. Вместо этого он создает объект `Executor` для выполнения фоновой работы всех экземпляров `AsyncTask` в одном фоновом потоке. Это означает, что экземпляры `AsyncTask` будут выполняться друг за другом, а затянувшаяся операция `AsyncTask` не позволит другим экземплярам `AsyncTask` получить процессорное время.

Организовать безопасное параллельное выполнение `AsyncTask` с использованием пула потоков возможно, но мы не рекомендуем так поступать. Если вы рассматриваете такое решение, обычно лучше самостоятельно организовать многопоточное выполнение, используя объекты `Handler` для взаимодействия с главным потоком, когда возникнет такая необходимость.

Упражнение. Предварительная загрузка и кэширование

Пользователи понимают, что не все происходит мгновенно (или по крайней мере большинство пользователей). Но несмотря на это, программисты стремятся к совершенству.

Для достижения моментального отклика в большинстве реальных приложений приведенный код расширяется в двух направлениях:

- добавление уровня кэширования;
- предварительная загрузка изображений.

Кэш представляет собой место для хранения определенного количества объектов `Bitmap`, чтобы они оставались в памяти даже после завершения использования. Объем кэша ограничен, поэтому вам понадобится стратегия выбора сохраняемых объектов при исчерпании свободного места. Многие кэши используют стратегию LRU (Least Recently Used): при нехватке свободного места из кэша удаляется элемент, который дольше всего не использовался.

Библиотека поддержки Android содержит класс с именем `LruCache`, реализующий стратегию LRU. В первом упражнении используйте `LruCache` для добавления простейшего кэширования `ThumbnailDownloader`. Каждый раз, когда для URL-адреса загружается объект `Bitmap`, вы помещаете его в кэш. Затем, когда требуется загрузить новое изображение, вы сначала проверяете содержимое кэша и смотрите, нет ли его в кэше.

После того как в программе будет создан кэш, он может использоваться для предварительной загрузки, то есть загрузки данных в кэш еще до того, как они фактически потребуются программе. Тем самым предотвращается задержка для загрузки объектов `Bitmap` до их вывода.

Качественно реализовать предварительную загрузку непросто, но она существенно меняет восприятие приложения пользователем. Во втором, более сложном упражнении для каждого выводимого элемента `GalleryItem` выполните предварительную загрузку 10 предшествующих и 10 следующих элементов `GalleryItem`.

28 Поиск

Следующим шагом в работе над приложением PhotoGallery станет поиск фотографий на Flickr. В этой главе вы узнаете, как правильно интегрировать поиск в приложение по правилам Android.

Впрочем, как выясняется, правильных способов несколько. Поиск был интегрирован в Android с самого начала, но он (как и кнопка меню) с тех пор заметно изменился. Как и меню, новый код поиска строился на базе существующих API. Таким образом, строя поиск в традиционном стиле, вы на самом деле готовитесь реализовать полноценный современный поиск Jelly Bean.

Поиск в Flickr

Начнем с того, что нужно сделать на стороне Flickr. Для выполнения поиска в Flickr следует вызвать метод `flickr.photos.search`. Вот как выглядит вызов метода `flickr.photos.search` для поиска текста «red»:

```
http://api.flickr.com/services/rest/?method=flickr.photos.search
&api_key=XXX&extras=url_s&text=red
```

Метод получает новые параметры, определяющие условия поиска, в частности параметр строки запроса. К счастью, разбор разметки XML, которая будет возвращена элементам `GalleryItem`, работает точно так же.

Внесите изменения, представленные в листинге 28.1, чтобы добавить в `FlickrFetchr` новый метод поиска. Поскольку разбор данных результатов `search` и `getRecent` для `GalleryItem` происходит одинаково, мы переработаем часть старого кода из `fetchItems()` в новый метод, который будет называться `downloadGalleryItems(String)`. Будьте внимательны — старый код `fetchItems()` перемещается в новую версию `fetchItems()`, а не удаляется из приложения.

Листинг 28.1. Добавление метода поиска фотографий Flickr (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "PhotoFetcher";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "4f721bbafa75bf6d2cb5af54f937bb70";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String METHOD_SEARCH = "flickr.photos.search";
    private static final String PARAM_EXTRAS = "extras";
    private static final String PARAM_TEXT = "text";
    ...

    public ArrayList<GalleryItem> fetchItems() {
    public ArrayList<GalleryItem> downloadGalleryItems(String  url) {
        ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();

        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                    .appendQueryParameter("method", METHOD_GET_RECENT)
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                    .build().toString();
            String xmlString = getUrl(url);
            Log.i(TAG, "Received xml: " + xmlString);
            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
            XmlPullParser parser = factory.newPullParser();
            parser.setInput(new StringReader(xmlString));

            parseItems(items, parser);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (XmlPullParserException xppe) {
            Log.e(TAG, "Failed to parse items", xppe);
        }
        return items;
    }

    public ArrayList<GalleryItem> fetchItems() {
        // Сюда перемещается код, приведенный выше
        String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_GET_RECENT)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .build().toString();
        return downloadGalleryItems(url);
    }

    public ArrayList<GalleryItem> search(String query) {
        String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_SEARCH)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .appendQueryParameter(PARAM_TEXT, query)
                .build().toString();
        return downloadGalleryItems(url);
    }
}
```

Метод downloadGalleryItems(String) используется дважды, потому что код загрузки и разбора XML одинаков для search и getRecent. Поиск сводится к простому вызову нового метода flickr.photos.search с передачей закодированной строки запроса в параметре text.

Теперь подключим тестовый код для вызова кода поиска из PhotoGalleryFragment.FetchItemsTask. Пока мы используем жестко запрограммированный запрос, чтобы убедиться в правильности работы поиска.

Листинг 28.2. Код с фиксированным запросом поиска (PhotoGalleryFragment.java)

```
    private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
        @Override
        protected ArrayList<GalleryItem> doInBackground(Void... params) {
            String query = "android"; // Только для тестирования

            if (query != null) {
                return new FlickrFetchr().search(query);
            } else {
                return new FlickrFetchr().fetchItems();
            }
        }

        @Override
        protected void onPostExecute(ArrayList<GalleryItem> items) {
            ...
        }
    }

    ...
}
```

По умолчанию используется старый код getRecent. Если поисковый запрос отличен от null (а теперь это условие выполняется всегда), то FetchItemsTask получает результаты поиска.

Запустите PhotoGallery и проверьте результаты. Если повезет, вы увидите пару фоток Энди.

Диалоговое окно поиска

В этом разделе мы реализуем в PhotoGallery поисковый интерфейс Android. Начнем с традиционного интерфейса диалогового окна.

Создание поискового интерфейса

В Honeycomb создатели Android избавились от физических кнопок поиска. Впрочем, даже до этого присутствие кнопки поиска не было гарантировано. Современные приложения Android с функцией поиска обязаны реализовать кнопку поиска, если они должны работать на устройствах до 3.0.

Реализация поиска не так уж сложна — достаточно вызвать метод Activity.onSearchRequested(). Этот метод выполняет точно такую же операцию, как и нажатие кнопки поиска.

Добавьте XML-файл в меню PhotoGallery res/menu/fragment_photo_gallery.xml. Нашему приложению также понадобится интерфейс для очистки условия поиска, поэтому добавьте кнопку сброса.

Листинг 28.3. Добавление команд меню поиска (res/menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    />
  <item android:id="@+id/menu_item_clear"
    android:title="@string/clear_search"
    android:icon="@android:drawable/ic_menu_close_clear_cancel"
    android:showAsAction="ifRoom"
    />
</menu>
```

Сейчас нам не хватает пары строк; добавьте их в strings.xml. (Позднее нам понадобится строка с подсказкой поиска, заодно добавим и ее.)

Листинг 28.4. Добавление строк поиска (res/values/strings.xml)

```
<resources>
    ...

    <string name="title_activity_photo_gallery">PhotoGalleryActivity</string>
    <string name="search_hint">Search Flickr</string>
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>

</resources>
```

Подключите обратные вызовы командного меню. Как сказано выше, для кнопки поиска будет вызываться метод `onSearchRequested()`. Для кнопки отмены пока не будет делаться ничего.

Листинг 28.5. Обратные вызовы командного меню (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);
    ...
}

...

@Override
public void onDestroyView() {
    ...
}
```

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            getActivity().onSearchRequested();
            return true;
        case R.id.menu_item_clear:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Опробуйте новый интерфейс меню и убедитесь в том, что он правильно отображается.

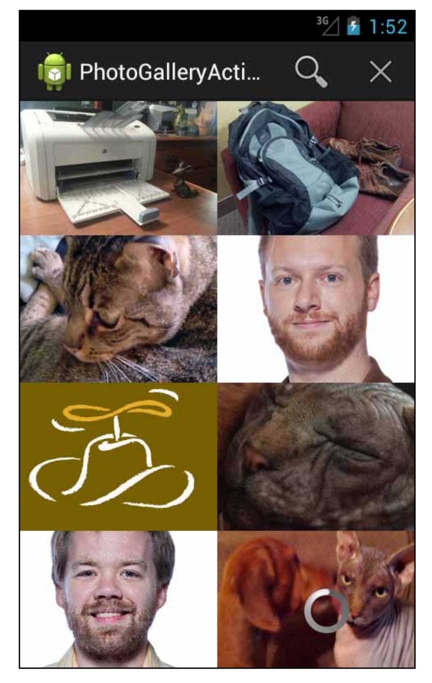


Рис. 28.1. Новый интерфейс поиска

Впрочем, при нажатии кнопки поиска сейчас ничего не происходит. Чтобы метод `onSearchRequested()` заработал, необходимо сделать `PhotoGalleryActivity` *поисковой активностью* (searchable activity).

Поисковые активности

Два компонента делают активность поисковой. Первый — файл XML. Он содержит элемент с именем `searchable`, который описывает, как должно отображаться диалоговое окно поиска. Создайте новую папку с именем res/xml, а в ней — новый файл XML с именем searchable.xml. Заполните файл простой версией элемента `searchable`.

Листинг 28.6. Конфигурация поиска (res/xml/searchable.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:hint="@string/search_hint"
  />
```

Эта разметка XML называется *конфигурацией поиска* (search configuration). В нашей ситуации конфигурация ограничивается строкой подсказки и именем приложения.

В более сложном приложении этот файл быстро разбухает. Рекомендации по поиску, голосовой поиск, глобальная конфигурация поиска, конфигурации действий, типы ввода — вся эта информация хранится в файле. Базовая конфигурация поиска необходима даже в самой простой реализации.

Следующий необходимый компонент находится в файле AndroidManifest.xml. Вы должны изменить *режим запуска* приложения, а также объявить дополнительный фильтр интентов и блок метаданных для `PhotoGalleryActivity`. Фильтр интентов сообщает о прослушивании поисковых интентов, а метаданные связывают только что написанную разметку XML с активностью.

Все эти объявления сообщают диспетчеру поиска Android, что ваша активность способна обрабатывать поисковые запросы, а также определяют ее конфигурацию поиска. *Диспетчер поиска* (SearchManager) — служба уровня ОС, отвечающая за вывод диалогового окна поиска и управление его взаимодействиями.

Откройте файл AndroidManifest.xml. Добавьте в него два элемента и атрибут `android:launchMode`, приведенные в листинге 28.7.

Листинг 28.7. Добавление фильтра интентов и метаданных (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  ...

  <application
    ... >
    <activity
      android:name=".PhotoGalleryActivity"
      android:launchMode="singleTop"
      android:label="@string/title_activity_photo_gallery" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
      <intent-filter>
```

```
        <action android:name="android.intent.action.SEARCH" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
        android:resource="@xml/searchable"/>
    </activity>
  </application>

</manifest>
```

Для начала разберемся с первыми двумя элементами. Первое добавление — уже знакомое определение фильтра интентов. Результаты поиска передаются посредством вызова `startActivity(…)` с интентом, которому назначено действие `action.intent.action.SEARCH`. Поисковый запрос включается в интент как дополнение. Таким образом, чтобы показать, что активность может обрабатывать результаты поиска, вы определяете фильтр для интента `action.intent.action.SEARCH`.

Второе добавление — тег метаданных. Мы уже использовали метаданные в главе 16, но этот тег выглядит иначе. Вместо атрибута `android:value` он использует `android:resource`. Следующий пример демонстрирует различия между двумя формами. Допустим, вы ссылаетесь на один строковый ресурс в двух разных тегах метаданных:

```
<meta-data android:name="metadata.value"
  android:value="@string/app_name" />
<meta-data android:name="metadata.resource"
  android:resource="@string/app_name" />
```

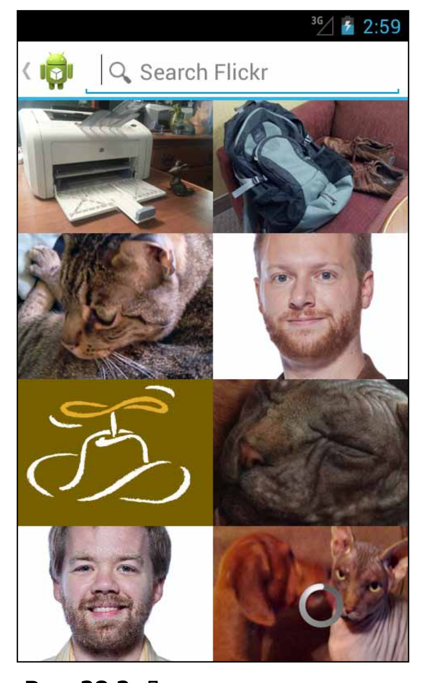


Рис. 28.2. Диалоговое окно поиска

Обратившись к значению `metadata.value`, мы бы обнаружили, что в нем содержится строка «PhotoGallery» — значение, хранящееся в `@string/app_name`. Однако значением `metadata.resource` будет целочисленный идентификатор этого ресурса. Другими словами, значение `metadata.resource` соответствует значению `R.string.app_name` в коде.

Вернемся от теории к практике. `SearchManager` должен передаваться целочисленный идентификатор searchable.xml, а не строковое значение этого файла XML; соответственно мы используем `android:resource` и передаем `SearchManager` идентификатор ресурса этого файла.

Как насчет атрибута `android:launchMode` в теге `activity`? Он определяет *режим запуска* активности. Вскоре мы расскажем о нем подробнее, когда будем писать код получения поискового запроса.

После выполнения всей необходимой подготовки мы можем открыть диалоговое окно поиска. Запустите приложение PhotoGallery и нажмите кнопку поиска в меню.

Аппаратная кнопка поиска

Тот же код, который выполняется при ручном вызове `onSearchRequested()`, будет выполняться при нажатии аппаратной кнопки поиска на старых устройствах. Если вы хотите самостоятельно убедиться в этом, включите в любой эмулятор до версии 3.0 аппаратную кнопку поиска; для этого следует настроить эмулятор на использование аппаратной клавиатуры, как показано на рис. 28.3.

Рис. 28.3. Добавление поддержки аппаратной клавиатуры

Как работает поиск

Поиск в Android базируется на кратко упоминавшейся ранее концепции *поисковой активности*. Поисковая активность определяется двумя компонентами: поисковым фильтром интентов и поисковыми метаданными конфигурации поиска.

С аппаратной кнопкой поиска каждое поисковое взаимодействие до самого интента поиска обрабатывается системой. Она обращается к файлу AndroidManifest.xml и проверяет, является ли активность поисковой. Если проверка дает положительный результат, то система выводит активность диалогового окна поиска поверх вашей активности. Эта активность запускает поиск, отправляя новый интент.

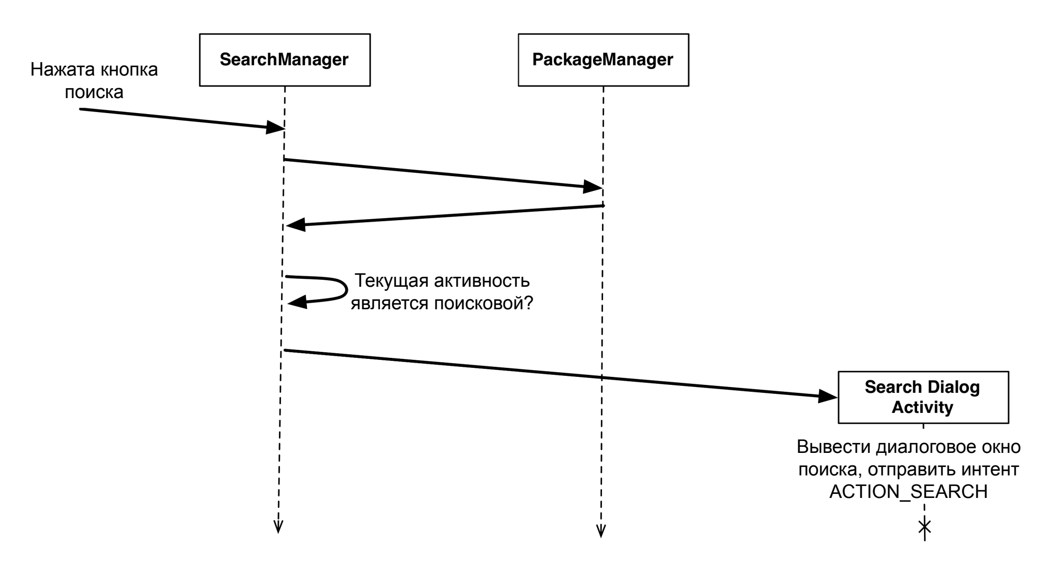


Рис. 28.4. Системный поиск

Это означает, что обычно при нажатии кнопки поиска запускается новая активность. Однако в нашем случае этого не происходит. Почему? Потому что мы добавили атрибут `android:launchMode="singleTop"` (листинг 28.7), изменяющий режим запуска.

Режимы запуска и новые интенты

Что такое «режим запуска» (launch mode)? Режимы запуска определяют, как ваша активность должна запускаться при получении нового интента, а в некоторых ситуациях — как она должна вести себя при запуске интента для запуска другой активности.

Таблица 28.1. Режимы запуска

Режим запуска	Поведение
standard	Поведение по умолчанию — для каждого полученного интента запускается новая активность
singleTop	Если экземпляр этой активности уже находится в верхней позиции стека возврата, то новый интент передается существующей активности вместо создания новой
singleTask	Активность запускается в отдельной задаче. Если активность уже существует в задаче, то все активности, находящиеся выше нее в стеке возврата, сбрасываются, а новый интент передается существующей активности
singleInstance	Активность запускается в отдельной задаче. Это единственная активность в своей задаче — если из задачи запускаются другие активности, они будут запущены в отдельных задачах. Если активность уже существует, то новый интент передается существующей активности

Все активности, написанные нами до настоящего момента, использовали стандартный режим запуска. Это поведение нам хорошо знакомо: когда интент преобразуется в активность со стандартным режимом запуска, новый экземпляр этой активности создается и добавляется в стек возврата.

Для `PhotoGalleryActivity` такое поведение не подходит (причины будут объяснены позднее, когда мы будем рассматривать `SearchView` и Honeycomb). Вместо него приходится задавать режим запуска `singleTop`.

Это означает, что вместо запуска новой активности полученный вами поисковый интент будет передаваться уже выполняемой активности `PhotoGalleryActivity` на вершине стека возврата.

Для получения нового интента вы переопределяете метод `onNewIntent(Intent)` в `Activity`. При каждом получении нового интента следует обновлять элементы `PhotoGalleryFragment`.

Переработайте `PhotoGalleryFragment` и включите метод `updateItems()`, который запускает `FetchItemsTask` для обновления текущих элементов.

Листинг 28.8. Добавление метода обновления элементов (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);

    new FetchItemsTask().execute();
    updateItems();

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    mThumbnailThread.setListener(new ThumbnailDownloader.Listener<ImageView>() {
        ...
    });
    mThumbnailThread.start();
    mThumbnailThread.getLooper();
}
```

```
public void updateItems() {
    new FetchItemsTask().execute();
}
```

Затем добавьте в PhotoGalleryActivity переопределение onNewIntent(Intent), которое переопределяет новый интент и обновляет элементы PhotoGallery-Fragment:

Листинг 28.9. Переопределение onNewIntent(...) (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends SingleFragmentActivity {
    private static final String TAG = "PhotoGalleryActivity";

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }

    @Override
    public void onNewIntent(Intent intent) {
        PhotoGalleryFragment fragment = (PhotoGalleryFragment)
            getSupportFragmentManager().findFragmentById(R.id.fragmentContainer);

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUERY);
            Log.i(TAG, "Received a new search query: " + query);
        }

        fragment.updateItems();
    }
}
```

Теперь при проведении поиска в LogCat должна появляться информация о получении PhotoGalleryActivity нового интента. Все элементы PhotoGallery сначала возвращаются к изображению лица Брайана, а затем обновляются.

Одно важное замечание по поводу onNewIntent(Intent): если вам понадобится значение нового интента, сохраните его где-нибудь. В значении, получаемом от getIntent(), будет содержаться старый интент, а не новый. Дело в том, что метод getIntent() предназначен для возвращения интента, запустившего эту активность, а не последнего полученного ей интента.

Следующим шагом должна стать интеграция поискового запроса в ваше приложение. В нашей реализации поиска в любой момент времени будет существовать только один поисковый запрос. Было бы удобно реализовать долгосрочное хранение этого запроса...

Простое сохранение с использованием механизма общих настроек

Для сохранения данных можно воспользоваться сериализацией объектов в флэш-память, как было сделано в главе 17. Однако для простых значений механизм общих настроек проще реализуется и удобнее в использовании.

Хранилище общих настроек (shared preferences) представляет собой файлы в файловой системе, для чтения и редактирования которых используется класс SharedPreferences. Экземпляр SharedPreferences работает как хранилище пар «ключ-значение», имеющее много общего с Bundle, но с возможностью долгосрочного хранения. Ключами являются строки, а значениями — атомарные типы данных. При ближайшем рассмотрении выясняется, что файлы содержат простую разметку XML, но благодаря классу SharedPreferences на эту подробность реализации можно не обращать внимания.

Чтобы приступить к использованию простого строкового значения в общих настройках, следует определить константу, которая станет ключом для вашего значения. Включите константу в FlickrFetchr.

Листинг 28.10. Константа общих настроек (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    public static final String PREF_SEARCH_QUERY = "searchQuery";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    ...
```

Для получения конкретного экземпляра SharedPreferences можно воспользоваться методом Context.getSharedPreferences(String,int). Однако на практике часто важен не конкретный экземпляр, а его совместное использование в пределах всего приложения. В таких ситуациях лучше использовать метод PreferenceManager.getDefaultSharedPreferences(Context), который возвращает экземпляр с именем по умолчанию и закрытыми (private) разрешениями.

В классе PhotoGalleryActivity получите объект SharedPreferences по умолчанию и сохраните запрос.

Листинг 28.11. Сохранение поискового запроса (PhotoGalleryActivity.java)

```
@Override
public void onNewIntent(Intent intent) {
    PhotoGalleryFragment fragment = (PhotoGalleryFragment)
getSupportFragmentManager()
            .findFragmentById(R.id.fragmentContainer);

    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
        String query = intent.getStringExtra(SearchManager.QUERY);
        Log.i(TAG, "Received a new search query: " + query);

        PreferenceManager.getDefaultSharedPreferences(this)
            .edit()
            .putString(FlickrFetchr.PREF_SEARCH_QUERY, query)
            .commit();
    }
    fragment.updateItems();
}
```

В приведенном выше коде мы вызываем SharedPreferences.edit() для получения экземпляра SharedPreferences.Editor. Мы используем этот класс для сохранения

значений в SharedPreferences. Он позволяет группировать изменения в транзакциях, по аналогии с тем, как это делалось с FragmentTransaction: множественные изменения группируются вместе для сохранения одной операцией записи.

После того как все изменения будут внесены, вызов commit() для объекта Editor делает их видимыми для других пользователей файла SharedPreferences.

Получение ранее сохраненного значения сводится к простому вызову SharedPreferences.getString(…), getInt(…) или другого метода, соответствующего типу данных. Добавьте в PhotoGalleryFragment код выборки поискового запроса из объекта SharedPreferences по умолчанию.

Листинг 28.12. Чтение сохраненного поискового запроса (PhotoGalleryFragment.java)

```
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
        String query = "android"; // just for testing
        Activity activity = getActivity();
        if (activity == null)
            return new ArrayList<GalleryItem>();

        String query = PreferenceManager.getDefaultSharedPreferences(activity)
            .getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
        if (query != null) {
            return new FlickrFetchr().search(query);
        } else {
            return new FlickrFetchr().fetchItems();
        }
    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> items) {
        ...
    }
}
```

Сохранять значения в общих настройках намного проще, чем сериализовать данные в JSON, не так ли?

Теперь поиск должен работать. Запустите приложение PhotoGallery, попробуйте что-нибудь поискать и посмотрите, что из этого выйдет.

Чтобы реализовать отмену поиска, удалите условие поиска из общих настроек и снова вызовите updateItems().

Листинг 28.13. Реализация отмены (PhotoGalleryFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_clear:
            PreferenceManager.getDefaultSharedPreferences(getActivity())
                .edit()
                .putString(FlickrFetchr.PREF_SEARCH_QUERY, null)
                .commit();
```

продолжение ⇗

```
            updateItems();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Использование SearchView в Android версий выше 3.0

Построенный нами поисковый интерфейс работает где угодно. Тем не менее его работа не соответствует рекомендациям для Honeycomb.

В Honeycomb добавился новый класс с именем `SearchView`. `SearchView` является *представлением действия* (action view) — то есть представлением, которое может включаться на панель действий. `SearchView` позволяет переместить весь поисковый интерфейс на панель действий активности (вместо использования диалогового окна, накладываемого на активность). В этом случае поисковый интерфейс использует то же стилевое оформление и тему, как ваше приложение, что безусловно хорошо.

Использование представления действия сводится к простому добавлению атрибута `android:actionViewClass` в тег элемента меню.

Листинг 28.14. Добавление представления действия в меню (res/menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    android:actionViewClass="android.widget.SearchView"
    />
  <item android:id="@+id/menu_item_clear"
    ...
    />
</menu>
```

Определяя представление действия, вы фактически говорите: «Android, вместо обычного представления для этого элемента панели действия нужно использовать этот класс представления». Обычно это также означает изменения в поведении. Например, `SearchView` не генерирует обратные вызовы `onOptionsItemSelected(…)`. И это хорошо, потому что вы можете оставить эти обратные вызовы для старых устройств, не поддерживающих представления действий.

(Раз уж речь зашла о старых устройствах, возможно, вы заметили класс `SearchViewCompat` в библиотеке поддержки. К сожалению, имя обманчиво — `SearchViewCompat` не является версией `SearchView`, которая может использоваться на старых устройствах. Вместо этого класс содержит пару статических методов, упрощающих избирательную вставку `SearchView` там, где эта функциональность доступна. Нашу проблему этот класс не решает.)

Если хотите, запустите PhotoGallery и посмотрите, как выглядит представление SearchView. Впрочем, ничего делать оно не будет. Прежде чем SearchView начнет отправлять поисковые интенты, ему необходимо знать вашу конфигурацию поиска. Необходимо добавить в onCreateOptionsMenu(…) код, который извлекает конфигурацию поиска и отправляет ее SearchView.

Благодаря системной службе SearchManager это не так сложно, как может показаться. У SearchManager имеется метод getSearchableInfo(ComponentName), который проверяет манифест, упаковывает всю актуальную информацию и возвращает ее в виде объекта SearchableInfo. Далее остается лишь передать объект SearchableInfo экземпляру SearchView. Для этого используется код, приведенный в листинге 28.15.

Листинг 28.15. Настройка SearchView (PhotoGalleryFragment.java)

```
@Override
@TargetApi(11)
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // Получение SearchView
        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        SearchView searchView = (SearchView)searchItem.getActionView();

        // Получение данных из файла searchable.xml
        // в виде объекта SearchableInfo
        SearchManager searchManager = (SearchManager)getActivity()
            .getSystemService(Context.SEARCH_SERVICE);
        ComponentName name = getActivity().getComponentName();
        SearchableInfo searchInfo = searchManager.getSearchableInfo(name);

        searchView.setSearchableInfo(searchInfo);
    }
}
```

Все начинается с поиска SearchView. Для этого мы ищем поисковый элемент MenuItem по идентификатору, а затем получаем его представление действия вызовом getActionView().

Затем у SearchManager запрашивается конфигурация поиска. SearchManager — системная служба, отвечающая за все, что относится к поиску. Ранее именно объект SearchManager действовал незаметно для нас, извлекая конфигурацию поиска и отображая поисковый интерфейс. Вся информация о поиске, включающая имя активности, которая должна получить интент, и все остальное из searchable.xml, хранится в объекте SearchableInfo, который мы получаем вызовом getSearchableInfo(ComponentName).

После получения объекта SearchableInfo мы сообщаем о нем SearchView вызовом setSearchableInfo(SearchableInfo). В результате экземпляр SearchView полностью связан с приложением. Запустите приложение на устройстве с версией выше 3.0 и посмотрите, как оно работает.

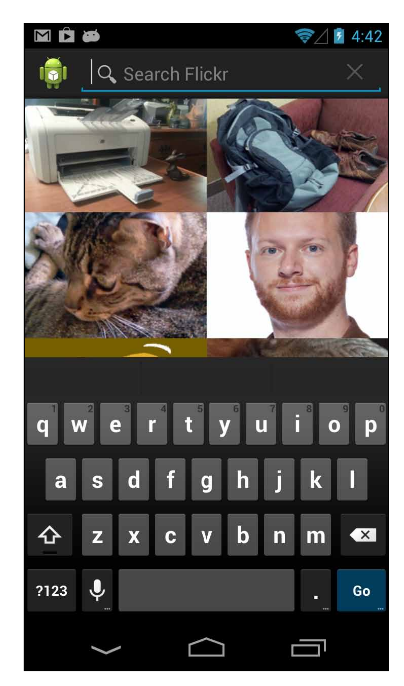


Рис. 28.5. Интеграция поиска в активность приложения

После правильной настройки `SearchView` ведет себя точно так же, как наши предыдущие реализации поиска.

Кроме одной маленькой подробности: если вы попытаетесь использовать аппаратную клавиатуру в эмуляторе, поиск выполняется два раза, один за другим.

Оказывается, в `SearchView` имеется ошибка. Нажатие клавиши ввода на аппаратной клавиатуре приводит к двукратному инициированию интента поиска. В активностях с режимом запуска по умолчанию это приводит к запуску двух одинаковых активностей для одного поиска.

Мы уже подготовились к этой ошибке ранее, когда выбрали режим запуска `singleTop`. Это гарантирует, что интенты будут сначала отправляться существующей активности, поэтому при отправке дубликата поискового интента новая активность запускаться не будет. К сожалению, из-за дубликата поиск все равно будет выполняться два раза подряд, но это гораздо лучше запуска двух одинаковых активностей для одного поиска.

Упражнения

Упражнения этой главы не слишком сложны. Первое заключается в использовании метода `Activity.startSearch(…)`.

Во внутренней реализации `onSearchRequested()` вызывает `Activity.startSearch(…)` — более детализированный способ запуска диалогового окна поиска.

Метод startSearch(…) позволяет задать исходный запрос, отображаемый в поле EditText, добавить объект Bundle с данными, отправляемыми поисковой активности-получателю, в дополнениях интентов, или запросить глобальное диалоговое окно веб-поиска (аналогичное тому, которое вы увидите при нажатии кнопки поиска на домашнем экране).

В первом небольшом упражнении используйте метод Activity.startSearch(…) для заполнения диалогового окна поиска текущим запросом и его выделения.

Второе упражнение требует, чтобы при запуске нового поиска выводилось общее количество доступных результатов поиска в оповещении Toast. Для этого вам придется обратиться к разметке XML, полученной от Flickr. В ней присутствует атрибут верхнего уровня с количеством возвращенных результатов.

29 Фоновые службы

Весь код, написанный нами до настоящего момента, был связан с активностью; это подразумевало, что он связывается с некой информацией на экране, видимой пользователю.

А если приложение не использует экран? Что, если выполняемые им операции не требуют визуального представления — как, скажем, воспроизведение музыки или проверка новых сообщений в блогах из поставки RSS? Для таких целей следует создать *службу* (service).

В этой главе мы добавим в PhotoGallery новую функцию фонового оповещения о появлении новых результатов поиска. Каждый раз, когда становится доступным новый результат, пользователь получает уведомление на панели состояния.

Создание IntentService

Начнем с создания службы. В этой главе мы будем использовать класс `IntentService`. Это не единственная разновидность служб, но, пожалуй, самая распространенная. Создайте субкласс `IntentService` с именем `PollService`. Эта служба будет использоваться нами для опроса результатов поиска.

Заглушка метода `onHandleIntent(Intent)` класса `PollService` будет сгенерирована автоматически. Включите в `onHandleIntent(Intent)` команду регистрации в журнале, добавьте тег для журнала и определите конструктор по умолчанию.

Листинг 29.1. Создание класса PollService (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

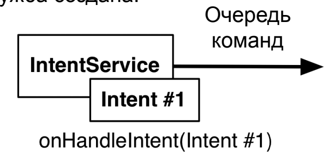
    public PollService() {
        super(TAG);
    }
```

```
    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i(TAG, "Received an intent: " + intent);
    }
}
```

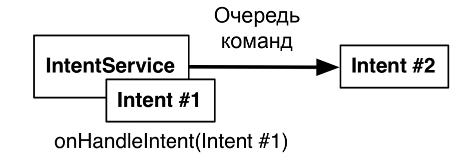
Это очень простая реализация `IntentService`. Что она делает? В общем-то она отчасти похожа на активность. Она является контекстом (`Service` — субкласс `Context`) и реагирует на интенты (как видно из `onHandleIntent(Intent)`).

Интенты службы называются *командами* (commands). Каждая команда представляет собой инструкцию по выполнению некоторой операции для службы. Способ обработки команды зависит от вида службы.

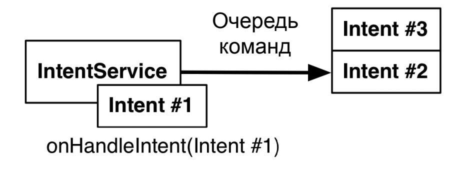
1. Получен командный интент 1.
Служба создана.



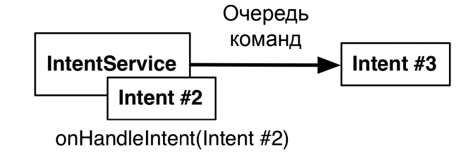
2. Получен командный интент 2.



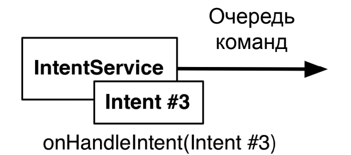
3. Получен командный интент 3.



4. Получен командный интент 1.



5. Командный интент 2 завершен.



6. Командный интент 3 завершен.
Служба уничтожена.

Рис. 29.1. Как IntentService обслуживает команды

`IntentService` обрабатывает команды, организованные в очередь. При получении первой команды `IntentService` инициализируется, запускает фоновый поток и помещает команду в очередь.

Затем `IntentService` переходит к последовательной обработке команд, с вызовом метода `onHandleIntent(Intent)` своего фонового потока для каждой команды. Новые

поступающие команды ставятся в очередь. Когда в очереди не остается ни одной команды, служба останавливается и уничтожается.

Приведенное описание относится только к `IntentService`. Позднее в этой главе службы и принципы обработки команд будут рассмотрены в более широкой перспективе.

Из того, что вы только что узнали о работе `IntentService`, возникает предположение, что службы реагируют на интенты. И это действительно так! А поскольку службы, как и активности, реагируют на интенты, они также должны объявляться в файле AndroidManifest.xml. Добавьте в манифест элемент для службы `PollService`.

Листинг 29.2. Добавление службы в манифест (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  ...

  <application
    ... >
    <activity
      android:name=".PhotoGalleryActivity"
      ... >
      ...
    </activity>
    <service android:name=".PollService" />
  </application>

</manifest>
```

Добавьте код запуска службы в `PhotoGalleryFragment`.

Листинг 29.3. Добавление кода запуска службы (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}
```

Запустите приложение и посмотрите, что получилось (рис. 29.2).

```
D  12-01 19:45…  jdwp          Got wake-up signal, bailing out of select
D  12-01 19:45…  dalvikvm      Debugger has detached; object registry had 1 entries
I  12-01 19:45…  PhotoFetcher  Fetching URL: http://api.flickr.com/services/rest/?method=flickr.phot
                               bbafa75bf6d2cb5af54f937bb70&extras=url_s
I  12-01 19:45…  PollService   Received an intent: Intent { cmp=com.bignerdranch.android.photogaller
```

Рис. 29.2. Первые шаги нашего приложения

Зачем нужны службы

Ладно, признаем: все эти записи LogCat выглядят скучно. Но сам-то код очень интересный! Почему? Что с ним можно сделать?

Пора вернуться в вымышленный мир, где мы уже не программисты, а хозяева обувного магазина с продавцами-супергероями.

Продавцы могут работать в двух местах: в торговом зале, где они общаются с покупателями, и на складе, куда покупатели не заходят. Склад может быть большим или маленьким в зависимости от магазина.

До настоящего момента весь наш код выполнялся в активностях. Активности — «прилавок» приложений Android. Весь этот код направлен на то, чтобы обеспечить приятные визуальные впечатления у пользователя.

Службы — своего рода «склад» приложений Android. Здесь происходит то, о чем пользователю знать не обязательно. Работа здесь может продолжаться после того, как торговый зал будет закрыт, когда активности давно перестали существовать.

Впрочем, довольно о магазинах. Что такого можно сделать со службой, чего нельзя сделать с активностью? Например, службу можно запустить, пока пользователь занимается другими делами.

Безопасные сетевые операции в фоновом режиме

Наша служба будет опрашивать Flickr в фоновом режиме. Чтобы выполнение сетевых операций в фоновом режиме проходило безопасно, потребуется дополнительный код. Android дает пользователю возможность отключить сетевые операции для фоновых приложений. Если вы используете множество приложений, интенсивно использующих вычислительные ресурсы, это может существенно повысить производительность.

Однако это означает, что при выполнении операций в фоновом режиме необходимо при помощи объекта `ConnectivityManager` убедиться в том, что сеть доступна. Поскольку исторически API изменялся, для этого потребуются две проверки, а не одна. Первая проверяет истинность `ConnectivityManager.getBackgroundDataSetting()`, а вторая — что результат `ConnectivityManager.getActiveNetworkInfo()` отличен от `null`.

Добавьте код из листинга 29.4 для выполнения этих проверок, а потом займемся техническими подробностями.

Листинг 29.4. Проверка доступности сети для фоновых операций (PollService.java)

```
@Override
public void onHandleIntent(Intent intent) {
    ConnectivityManager cm = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
    @SuppressWarnings("deprecation")
    boolean isNetworkAvailable = cm.getBackgroundDataSetting() &&
        cm.getActiveNetworkInfo() != null;
    if (!isNetworkAvailable) return;

    Log.i(TAG, "Received an intent: " + intent);
}
```

Почему нужны две проверки? В старых версиях Android полагалось проверять `getBackgroundDataSetting()` и прерывать операцию, если вызов возвращал `false`. Если проверка не выполнялась, приложение преспокойно использовало сетевые данные. Такая схема была слишком несовершенной: программист мог случайно забыть о проверке, а то и проигнорировать ее.

В Android 4.0, Ice Cream Sandwich, схема была изменена: при запрете фоновых операций сеть просто блокировалась. Вот почему мы проверяем, не возвращает ли `getActiveNetworkInfo()` значение `null`. Это правильно с точки зрения пользователя, потому что смысл режима фоновых операций всегда соответствует его ожиданиям. Конечно, это немного усложняет жизнь разработчика.

Чтобы использовать метод `getActiveNetworkInfo()`, также необходимо получить разрешение `ACCESS_NETWORK_STATE`.

Листинг 29.5. Получение разрешения для проверки состояния сети (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
  </application>

</manifest>
```

Поиск новых результатов

Наша служба будет опрашивать Flickr на появление новых результатов, поэтому ей нужно знать результат последней выборки. Для этой работы идеально подойдет механизм `SharedPreferences`. Добавьте в `FlickrFetchr` еще одну константу для хранения идентификатора последней загруженной фотографии.

Листинг 29.6. Добавление константы для хранения идентификатора (FlickrFetchr.java)

```
public class FlickrFetchr {
    public static final String TAG = "PhotoFetcher";

    public static final String PREF_SEARCH_QUERY = "searchQuery";
    public static final String PREF_LAST_RESULT_ID = "lastResultId";
```

```
    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "xxx";
    ...
```

Следующим шагом станет заполнение кода службы. Необходимо сделать следующее:

1. Прочитать текущий запрос и идентификатор последнего результата из SharedPreferences.
2. Загрузить последний набор результатов с использованием `FlickrFetchr`.
3. Если набор не пуст, получить первый результат.
4. Проверить, отличается ли его идентификатор от идентификатора последнего результата.
5. Сохранить первый результат в `SharedPreferences`.

Давайте вернемся к файлу PollService.java и претворим этот план в жизнь. В листинге 29.7 содержится довольно длинный блок кода, но в нем нет ничего, что бы вы не видели ранее.

Листинг 29.7. Проверка новых результатов (PollService.java)

```
@Override
protected void onHandleIntent(Intent intent) {
    ...

    if (!isNetworkAvailable) return;

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
    String query = prefs.getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
    String lastResultId = prefs.getString(FlickrFetchr.PREF_LAST_RESULT_ID, null);

    ArrayList<GalleryItem> items;
    if (query != null) {
        items = new FlickrFetchr().search(query);
    } else {
        items = new FlickrFetchr().fetchItems();
    }

    if (items.size() == 0)
        return;

    String resultId = items.get(0).getId();

    if (!resultId.equals(lastResultId)) {
        Log.i(TAG, "Got a new result: " + resultId);
    } else {
        Log.i(TAG, "Got an old result: " + resultId);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}
```

Видите каждый шаг из приведенного списка? Хорошо. Запустите приложение PhotoGallery и вы увидите, как приложение получает исходные результаты. Если поисковый запрос уже выбран, вероятно, при последующих запусках будут отображаться устаревшие результаты.

Отложенное выполнение и AlarmManager

Чтобы служба реально использовалась в фоновом режиме, нам понадобится какой-то механизм организации операций при отсутствии работающих активностей — например, таймер, который срабатывает каждые пять минут, или что-нибудь в этом роде.

Это можно сделать при помощи `Handler`, вызовами методов `Handler.sendMessageDelayed(…)` или `Handler.postDelayed(…)`. Впрочем, такое решение с большой вероятностью перестанет работать, если пользователь уйдет со всех активностей. Процесс закроется, и вместе с ним прекратят существование сообщения `Handler`.

По этой причине вместо `Handler` мы будем использовать `AlarmManager` — системную службу, которая может отправлять интенты за вас.

Как сообщить `AlarmManager`, какие интенты нужно отправить? При помощи объекта `PendingIntent`. По сути, в объекте `PendingIntent` упаковывается пожелание: «Я хочу запустить `PollService`». Затем это пожелание отправляется другим компонентам системы, таким как `AlarmManager`.

Включите в `PollService` новый метод с именем `setServiceAlarm(Context,boolean)`, который включает и отключает сигнал за вас. Метод будет объявлен статическим; это делается для того, чтобы код сигнала размещался рядом с другим кодом `PollService`, с которым он связан, но мог вызываться другими компонентами. Обычно включение и отключение должно осуществляться из интерфейсного кода фрагмента или другого контроллера.

Листинг 29.8. Добавление сигнального метода (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 15; // 15 секунд

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...
    }

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = new Intent(context, PollService.class);
        PendingIntent pi = PendingIntent.getService(
                context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
```

```
                context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.RTC,
                    System.currentTimeMillis(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }
}
```

Метод начинается с создания объекта `PendingIntent`, который запускает `PollService`. Задача решается вызовом метода `PendingIntent.getService(…)`, в котором упаковывается вызов `Context.startService(Intent)`. Метод получает четыре параметра: `Context` для отправки интента; код запроса, по которому этот объект `PendingIntent` отличается от других; отправляемый объект `Intent`; и наконец, набор флагов, управляющий процессом создания `PendingIntent` (вскоре мы используем один из них).

После этого сигнал либо устанавливается, либо отменяется. Чтобы установить сигнал, следует вызвать `AlarmManager.setRepeating(…)`. Этот метод тоже получает четыре параметра: константу для описания временной базы сигнала (об этом чуть позже), время запуска сигнала, временной интервал повторения сигнала, и наконец, объект `PendingIntent`, запускаемый при срабатывании сигнала.

Отмена сигнала осуществляется вызовом `AlarmManager.cancel(PendingIntent)`. Обычно при этом также следует отменить и `PendingIntent`. Вскоре вы увидите, как отмена `PendingIntent` помогает в отслеживании статуса сигнала.

Добавьте простой тестовый код для запуска сигнала из `PhotoGalleryFragment`.

Листинг 29.9. Добавление кода запуска сигнала (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);
    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}
```

Введите этот код и запустите PhotoGallery. Немедленно нажмите кнопку Back и выйдите из приложения.

Замечаете что-нибудь в LogCat? `PollService` честно продолжает работать, запускаясь каждые 15 секунд. Для этого и нужен класс `AlarmManager`. Даже если процесс

будет завершен, AlarmManager будет выдавать интенты, снова и снова запуская PollService.

(Конечно, такое поведение в высшей степени возмутительно. Возможно, вам стоит удалить приложение, пока ситуация не будет исправлена.)

PendingIntent

Давайте поближе познакомимся с PendingIntent. PendingIntent представляет собой объект-маркер. Когда вы получаете такой объект вызовом PendingIntent.getService(…), вы тем самым говорите ОС: «Пожалуйста, запомни, что я хочу отправлять этот интент вызовом startService(Intent)». Позднее вы можете вызвать send() для PendingIntent, и ОС отправит изначально упакованный интент — точно так, как вы приказали.

А лучше всего здесь то, что если передать маркер PendingIntent другой стороне и эта сторона его использует, маркер будет отправлен *от имени вашего приложения*. А поскольку объект PendingIntent существует в ОС, вы сохраняете полный контроль над ним. Например, вы можете (просто из вредности) предоставить кому-нибудь объект PendingIntent и немедленно отменить его, так что вызов send() ничего не сделает.

Если вы запросите PendingIntent дважды с одним интентом, то получите тот же PendingIntent. Например, таким образом можно проверить существование PendingIntent или отменить ранее выданный объект PendingIntent.

Управление сигналами с использованием PendingIntent

Для каждого объекта PendingIntent можно зарегистрировать только один сигнал. Именно так работает вызов setServiceAlarm(boolean) при ложном значении isOn: он вызывает AlarmManager.cancel(PendingIntent) для отмены сигнала, связанного с вашим объектом PendingIntent, а потом отменяет PendingIntent.

Так как PendingIntent также удаляется при отмене сигнала, вы можете проверить, существует ли PendingIntent, чтобы узнать, активен сигнал или нет. Эта операция выполняется передачей флага PendingIntent.FLAG_NO_CREATE вызову PendingIntent.getService(…). Флаг говорит, что если объект PendingIntent не существует, то вместо его создания следует вернуть null.

Напишите новый метод isServiceAlarmOn(Context), использующий флаг PendingIntent.FLAG_NO_CREATE для проверки сигнала.

Листинг 29.10. Добавление метода isServiceAlarmOn() (PollService.java)

```
public static void setServiceAlarm(Context context, boolean isOn) {
    ...
}

public static boolean isServiceAlarmOn(Context context) {
    Intent i = new Intent(context, PollService.class);
    PendingIntent pi = PendingIntent.getService(
```

```
            context, 0, i, PendingIntent.FLAG_NO_CREATE);
    return pi != null;
}
```

Так как этот объект PendingIntent используется только для установки сигнала, null вместо PendingIntent означает, что сигнал не установлен.

Управление сигналом

Теперь, когда мы можем включать и отключать сигнал (а также определять, включен ли он), давайте добавим интерфейс для его включения и выключения. Добавьте в файл menu/fragment_photo_gallery.xml новую команду меню.

Листинг 29.11. Переключение режима опроса (menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/menu_item_search"
    android:title="@string/search"
    android:icon="@android:drawable/ic_menu_search"
    android:showAsAction="ifRoom"
    android:actionViewClass="android.widget.SearchView"
    />
  <item android:id="@+id/menu_item_clear"
    android:title="@string/clear_search"
    android:icon="@android:drawable/ic_menu_close_clear_cancel"
    android:showAsAction="ifRoom"
    />
  <item android:id="@+id/menu_item_toggle_polling"
    android:title="@string/start_polling"
    android:showAsAction="ifRoom"
    />
</menu>
```

Затем необходимо добавить несколько новых строк — одну для начала опроса, одну для завершения опроса. (Позднее нам понадобится еще пара строк для оповещений на панели состояния; добавим их сейчас.)

Листинг 29.12. Добавление строк для режима опроса (res/values/strings.xml)

```
<resources>
    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
    <string name="start_polling">Poll for new pictures</string>
    <string name="stop_polling">Stop polling</string>
    <string name="new_pictures_title">New PhotoGallery Pictures</string>
    <string name="new_pictures_text">You have new pictures in PhotoGallery.</
string>

</resources>
```

Удалите старый отладочный код для запуска сигнала и добавьте реализацию команды меню.

Листинг 29.13. Реализация команды переключения режима опроса (PhotoGalleryFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);
    updateItems();

    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}

...

@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            ...
        case R.id.menu_item_clear:
            ...

            updateItems();
            return true;
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Теперь вы сможете включать и отключать сигнал.

А как насчет команды меню?

Обновление элемента командного меню

Обычно командное меню достаточно просто заполнить. Однако в некоторых ситуациях требуется обновлять его элементы в соответствии с текущим состоянием приложения.

Командные меню не заполняются при каждом использовании, даже если это командные меню «старого стиля». Если вам потребуется обновить содержимое элемента командного меню, поместите соответствующий код в `onPrepareOptionsMenu(Menu)`. Этот метод вызывается при каждом определении конфигурации меню, а не только при его изначальном создании.

Добавьте реализацию onPrepareOptionsMenu(Menu), которая проверяет, установлен ли сигнал, а затем изменяет текст menu_item_toggle_polling, чтобы в меню отображался соответствующий текст.

Листинг 29.14. Добавление метода onPrepareOptionsMenu(Menu) (PhotoGalleryFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

@Override
public void onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);

    MenuItem toggleItem = menu.findItem(R.id.menu_item_toggle_polling);
    if (PollService.isServiceAlarmOn(getActivity())) {
        toggleItem.setTitle(R.string.stop_polling);
    } else {
        toggleItem.setTitle(R.string.start_polling);
    }
}
```

На устройствах с версией до 3.0 этот метод вызывается при каждом отображении меню. Это гарантирует, что элементы меню всегда будут содержать правильный текст. Если хотите удостовериться в этом, запустите приложение PhotoGallery в эмуляторе с версией до 3.0.

Однако для версий после 3.0 этого недостаточно. Панель действий не обновляется автоматически. Вам придется специально приказать ей вызвать onPrepareOptionsMenu(Menu) и обновить элементы вызовом Activity.invalidateOptionsMenu().

Добавьте следующий код в onOptionsItemSelected(MenuItem), чтобы приказать устройствам после 3.0 обновить свои панели действий.

Листинг 29.15. Командное меню объявляется недействительным (PhotoGalleryFragment.java)

```
@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
                getActivity().invalidateOptionsMenu();

            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

После этого код будет отлично работать в новом эмуляторе 4.2.

И все же в приложении кое-чего не хватает.

Оповещения

Служба успешно работает в фоновом режиме. Однако пользователь об этом понятия не имеет, так что пользы от нее будет немного.

Когда службе требуется передать какую-то информацию пользователю, для этого она почти всегда использует *оповещения* (notifications) — элементы, которые появляются на выдвижной панели оповещений, которую пользователь вызывает, проводя пальцем вниз от верха экрана.

Чтобы опубликовать оповещение, необходимо сначала создать объект `Notification`. Объекты `Notification` создаются с использованием объектов-построителей — подобно тому, как это делалось с `AlertDialog` из главы 12. Как минимум объект `Notification` должен иметь:

- *текст бегущей строки*, отображаемый на панели состояния при первом появлении оповещения;
- *значок*, отображаемый на панели состояния после исчезновения бегущей строки;
- *представление*, отображаемое на выдвижной панели оповещений для вывода оповещения;
- *объект PendingIntent*, срабатывающий при нажатии на оповещении на выдвижной панели.

После того как объект `Notification` будет создан, его можно отправить вызовом метода `notify(int, Notification)` для системной службы `NotificationManager`.

Чтобы служба `PollService` оповещала пользователя о появлении нового результата, добавьте код из листинга 29.16. Этот код создает объект `Notification` и вызывает `NotificationManager.notify(int, Notification)`.

Листинг 29.16. Добавление оповещения (PollService.java)

```
@Override
public void onHandleIntent(Intent intent) {
    ...

    String resultId = items.get(0).getId();

    if (!resultId.equals(lastResultId)) {
        Log.i(TAG, "Got a new result: " + resultId);

        Resources r = getResources();
        PendingIntent pi = PendingIntent
            .getActivity(this, 0, new Intent(this, PhotoGalleryActivity.class), 0);

        Notification notification = new NotificationCompat.Builder(this)
            .setTicker(r.getString(R.string.new_pictures_title))
            .setSmallIcon(android.R.drawable.ic_menu_report_image)
            .setContentTitle(r.getString(R.string.new_pictures_title))
```

```
            .setContentText(r.getString(R.string.new_pictures_text))
            .setContentIntent(pi)
            .setAutoCancel(true)
            .build();

        NotificationManager notificationManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);

        notificationManager.notify(0, notification);
    }
    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}
```

Пройдемся по этому коду сверху вниз. Сначала мы задаем текст бегущей строки и значок вызовами `setTicker(CharSequence)` и `setSmallIcon(int)`.

После этого задается внешний вид оповещения на самой выдвижной панели. Можно полностью определить внешний вид оповещения, но проще использовать стандартное оформление со значком, заголовком и текстовой областью. Для значка будет использоваться значение из `setSmallIcon(int)`. Заголовок и текст задаются вызовами `setContentTitle(CharSequence)` и `setContentText(CharSequence)` соответственно.

Теперь необходимо указать, что происходит при нажатии на оповещении. Как и в случае с `AlarmManager`, для этого используется `PendingIntent`. Объект `PendingIntent`, передаваемый `setContentIntent(PendingIntent)`, будет запускаться при нажатии пользователем на вашем оповещении на выдвижной панели. Вызов `setAutoCancel(true)` слегка изменяет это поведение: с этим вызовом оповещение при нажатии также будет удаляться с выдвижной панели оповещений.

Код завершается вызовом `NotificationManager.notify(…)`. Передаваемый целочисленный параметр содержит идентификатор оповещения, уникальный в границах приложения. Если вы отправите второе оповещение с тем же идентификатором, оно заменит последнее оповещение, отправленное с этим идентификатором. Так реализуются индикаторы прогресса и другие динамические визуальные эффекты.

Собственно, это все. Весь пользовательский интерфейс для фонового опроса! Когда вы убедитесь в том, что все работает правильно, замените константу интервала опроса более разумным значением.

Листинг 29.17. Изменение периодичности опроса (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static final int POLL_INTERVAL = 1000 * 15; // 15 секунд
    public static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 минут

    public PollService() {
        super(TAG);
    }
```

Для любознательных: подробнее о службах

Мы рекомендуем использовать `IntentService` для большинства реализаций служб. Если паттерн `IntentService` не подходит для вашей архитектуры, вам придется поближе познакомиться со службами для составления собственной реализации. Приготовьтесь, при изучении служб приходится учитывать множество подробностей и нюансов.

Что делают (и чего не делают) службы

Служба представляет собой компонент приложения, предоставляющий обратные вызовы жизненного цикла (в этом она похожа на активность). Эти обратные вызовы даже выполняются в главном потоке без каких-либо усилий с вашей стороны, как и в случае с активностью.

В своем исходном состоянии служба не выполняет никакой код в фоновом потоке. Это главная причина, по которой мы рекомендуем `IntentService`. Для большинства нетривиальных служб потребуется тот или иной фоновый поток, а `IntentService` автоматически управляет шаблонным кодом, необходимым для достижения этой цели.

Давайте посмотрим, какие обратные вызовы жизненного цикла предоставляет служба.

Жизненный цикл службы

Жизненный цикл службы, запущенной `startService(Intent)`, весьма прост. Ниже перечислены методы обратного вызова жизненного цикла.

- `onCreate(…)` — вызывается при создании службы.
- `onStartCommand(Intent,int,int)` — вызывается каждый раз, когда компонент запускает службу вызовом `startService(Intent)`. Два целочисленных параметра содержат набор флагов и идентификатор запуска. Флаги указывают, является ли интент повторной отправкой ранее доставленного интента, или повторной попыткой после неудачи при доставке. Идентификатор запуска отличается при разных вызовах `onStartCommand(Intent,int,int)`, поэтому по нему можно отличить эту команду от других.
- `onDestroy()` — вызывается, когда дальнейшее существование службы не требуется. Часто происходит после остановки службы.

Остается один вопрос: как происходит остановка службы? Это можно сделать разными способами в зависимости от типа службы. Тип службы определяется значением, возвращаемым методом `onStartCommand(…)`; возможные значения — `Service.START_NOT_STICKY`, `START_REDELIVER_INTENT` или `START_STICKY`.

Незакрепляемые службы

`IntentService` является *незакрепляемой* (non-sticky) службой, поэтому начнем с нее. Незакрепляемая служба останавливается, когда сама служба сообщает

о завершении своей работы. Чтобы сделать свою службу незакрепляемой, верните START_NOT_STICKY или START_REDELIVER_INTENT.

Чтобы сообщить Android о завершении работы службы, вызовите метод stopSelf() или stopSelf(int). Первый метод, stopSelf(), является безусловным. Он всегда останавливает службу независим от того, сколько раз был вызван метод onStartCommand(…).

IntentService использует вместо него stopSelf(int). Этот метод получает идентификатор запуска, полученный в onStartCommand(…). Служба останавливается только в том случае, если этот идентификатор является самым последним из полученных идентификаторов запуска (так работает внутренняя реализация IntentService).

Чем же отличаются START_NOT_STICKY и START_REDELIVER_INTENT? Поведением службы, если системе потребуется завершить ее преждевременно. Служба START_NOT_STICKY просто прекращает существование и уходит в никуда. Служба START_REDELIVER_INTENT, напротив, попытается запуститься позднее, когда ресурсы не будут так ограничены.

Выбор между START_NOT_STICKY и START_REDELIVER_INTENT определяется важностью этой операции для вашего приложения. Если служба не критична, выберите режим START_NOT_STICKY. В PhotoGallery служба запускается по сигналу. Пропажа одного вызова не критична, поэтому мы выбираем START_NOT_STICKY. Такое поведение используется по умолчанию для IntentService. Чтобы переключиться на режим START_REDELIVER_INTENT, вызовите IntentService.setIntentRedelivery(true).

Закрепляемые службы

Закрепляемая (sticky) служба остается запущенной, пока кто-то находящийся вне службы не прикажет ей остановиться, вызвав метод Context.stopService(Intent). Чтобы сделать службу закрепляемой, верните значение START_STICKY.

После запуска закрепляемая служба остается «включенной», пока компонент не вызовет Context.stopService(Intent). Если службу по какой-то причине потребуется уничтожить, она будет снова перезапущена с передачей onStartCommand(…) null-интента.

Закрепляемый режим хорошо подходит для долгоживущих служб (например, проигрывателя музыки), которые должны работать до тех пор, пока пользователь не прикажет им остановиться. Впрочем, даже в этом случае стоит рассмотреть альтернативную архитектуру с использованием незакрепляемых служб. Управлять закрепляемыми службами неудобно, потому что трудно определить, была ли уже запущена служба.

Привязка к службам

Также существует возможность привязки (binding) к службе при помощи метода bindService(Intent,ServiceConnection,int). *Привязка к службе* — механизм подключения к службе и непосредственного вызова ее методов. Привязка осуществляется

вызовом `bindService(Intent,ServiceConnection,int)`. `ServiceConnection` — объект, представляющий привязку к службе и получающий все обратные вызовы привязки.

В фрагменте код привязки выглядит примерно так:

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // Используется для взаимодействия со службой
        MyBinder binder = (MyBinder)service;
    }

    public void onServiceDisconnected(ComponentName className) {
    }
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent i = new Intent(c, MyService.class);
    c.bindService(i, mServiceConnection, 0);
}

@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().getApplicationContext().unbindService(mServiceConnection);
}
```

На стороне службы привязка определяет два дополнительных обратных вызова жизненного цикла:

- `onBind(Intent)` — вызывается каждый раз, когда создается привязка к службе. Возвращает объект `IBinder`, получаемый при вызове `ServiceConnection.onServiceConnected(ComponentName,IBinder)`.
- `onUnbind(Intent)` — вызывается при завершении привязки к службе.

Локальная привязка к службам

Что же собой представляет `MyBinder`? Если служба является локальной, это может быть простой объект Java, существующий в локальном процессе. Обычно он предоставляет дескриптор (handle), используемый для прямого вызова методов службы:

```
private class MyBinder extends IBinder {
    public MyService getService() {
        return MyService.this;
    }
}

@Override
public void onBind(Intent intent) {
    return new MyBinder();
}
```

Паттерн выглядит довольно заманчиво — это единственный механизм Android, позволяющий одному компоненту Android напрямую взаимодействовать с другим. Тем не менее мы не рекомендуем его использовать. Службы по сути являются синглетами, и такое их использование не предоставляет никаких заметных преимуществ перед использованием синглета.

Удаленная привязка к службе

Привязка приносит больше пользы для *удаленных служб*, потому что она дает возможность приложениям из других процессов вызывать методы вашей службы. Создание привязки к удаленной службе — нетривиальная тема, выходящая за рамки нашей книги. За подробностями обращайтесь к документации по AIDL или описанию класса `Messenger`.

30 Широковещательные интенты

На устройствах Android постоянно что-нибудь происходит. Точки WiFi входят и выходят из зоны приема, устанавливаются пакеты, поступают телефонные звонки и текстовые сообщения.

Если о возникновении некоторого события должны узнать многие компоненты системы, Android использует для распространения информации *широковещательные интенты* (broadcast intent). Широковещательные интенты работают примерно так же, как уже знакомые вам обычные интенты, не считая того, что их могут получать сразу несколько компонентов. Широковещательные интенты передаются *широковещательным приемникам* (broadcast receivers).

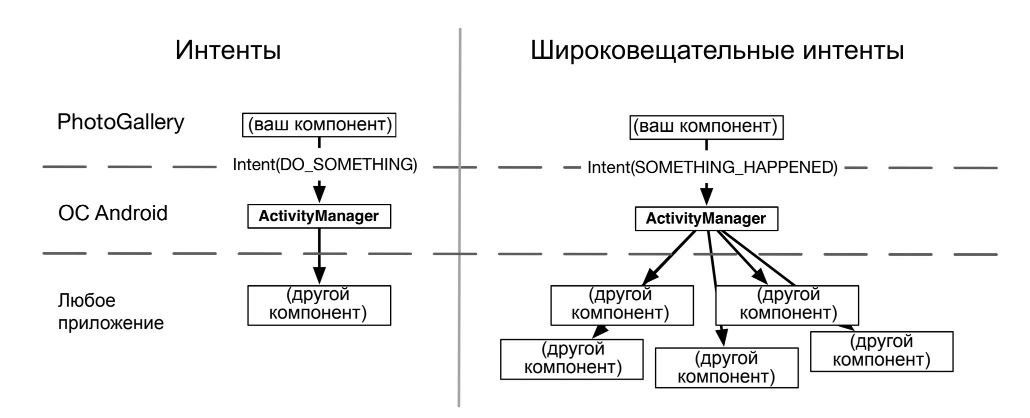


Рис. 30.1. Обычные и широковещательные интенты

В этой главе вы узнаете, как организовать прослушивание широковещательных интентов от системы, а также динамически отправлять и получать их в приложении

во время выполнения. Мы начнем с прослушивания широковещательной рассылки, сообщающей о загрузке устройства, а затем перейдем к отправке и получению ваших собственных широковещательных рассылок.

Пробуждение при загрузке

Фоновый сигнал PhotoGallery работает, но он не идеален. Если пользователь перезагрузит свой телефон, то сигнал будет потерян.

Приложения, выполняющие продолжительный процесс для пользователя, обычно должны пробуждаться после загрузки устройства. Чтобы узнать о завершении загрузки, следует прослушивать широковещательный интент с действием `BOOT_COMPLETED`.

Широковещательные приемники в манифесте

Давайте напишем широковещательный приемник. Начнем с создания нового класса Java с именем `StartupReceiver`, являющегося субклассом `android.content.BroadcastReceiver`. Eclipse реализует за вас один абстрактный метод с именем `onReceive(Context,Intent)`. Заполните код вашего класса следующим образом.

Листинг 30.1. Наш первый широковещательный приемник (StartupReceiver.java)

```
package com.bignerdranch.android.photogallery;
...

public class StartupReceiver extends BroadcastReceiver {
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());
    }
}
```

Широковещательный приемник — компонент, который получает интенты, как и служба или активность. И подобно службам и активностям, широковещательные приемники должны быть зарегистрированы в системе для выполнения любой полезной работы. Это не всегда означает, что они должны присутствовать в манифесте (впрочем, этот конкретный приемник в нем должен присутствовать).

Подключение приемника происходит точно так же, как и подключение службы или активности: вы используете тег `receiver` с соответствующими фильтрами интентов. `StartupReceiver` будет прослушивать действие `BOOT_COMPLETED`. Для этого действия также требуется разрешение, поэтому в манифест необходимо включить тег `uses-permission`.

Откройте файл AndroidManifest.xml и включите объявление `StartupReceiver`.

Листинг 30.2. Включение приемника в манифест (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

  <application
    ... >
    <activity
      ... >
      ...
    </activity>
    <service android:name=".PollService" />
    <receiver android:name=".StartupReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
  </application>

</manifest>
```

В отличие от активностей и служб, объявляемые в манифесте широковещательные приемники почти всегда объявляют фильтры интентов. Ведь широковещательные интенты предназначаются для отправки информации многим слушателям, тогда как у явных интентов приемник только один. В результате явные широковещательные интенты встречаются редко.

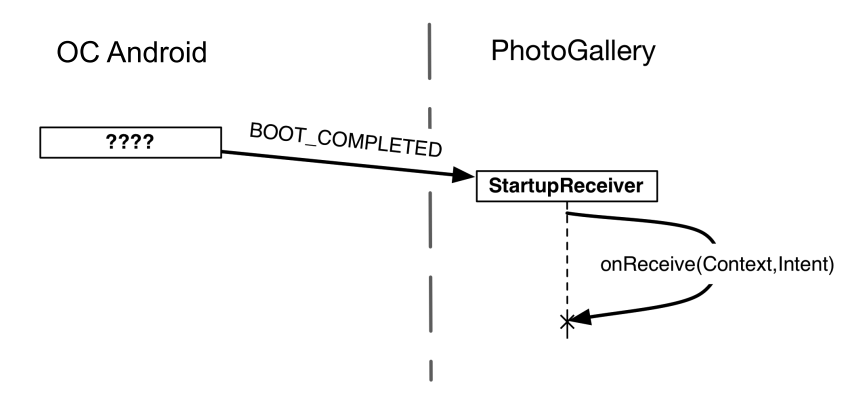


Рис. 30.2. Получение BOOT_COMPLETED

После того как широковещательный приемник был объявлен в вашем манифесте, он будет пробуждаться при каждой отправке соответствующего широковещательного интента — даже если ваше приложение в настоящий момент не выполняется. При пробуждении выполняется метод `onReceive(Context,Intent)` эфемерного широковещательного приемника, после чего он прекращает существовать.

Запустите приложение PhotoGallery, перезагрузите устройство или эмулятор и переключитесь на DDMS. Вы увидите в LogCat сообщение о запуске вашего приемника. Однако проверив устройство на вкладке Devices, вы, скорее всего, не найдете здесь процесс PhotoGallery. Этот процесс прожил ровно столько времени, сколько необходимо для выполнения широковещательного приемника, а потом снова перестал существовать.

Использование приемников

Тот факт, что широковещательные приемники живут такой короткой жизнью, ограничивает возможности их использования. Например, в них нельзя использовать асинхронные API или регистрировать слушателей, потому что срок жизни приемника не превысит продолжительности выполнения `onReceive(Context,Intent)`. Кроме того, `onReceive(Context,Intent)` выполняется в главном потоке, поэтому здесь нельзя выполнять никакие интенсивные вычисления — то есть никаких сетевых операций или серьезной работы с долговременным хранилищем.

Впрочем, это вовсе не значит, что приемники бесполезны. Они чрезвычайно полезны для выполнения всевозможного служебного кода. Наш сигнал должен сбрасываться при завершении запуска системы; эта задача достаточно мала, а широковещательный приемник идеально подходит для ее решения.

Приемник обязан знать, должен ли сигнал находиться во включенном или отключенном состоянии. Добавьте в `PollService` общую настройку, в которой будет храниться эта информация.

Листинг 30.3. Добавление настройки для хранения состояния сигнала (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public PollService() {
        super(TAG);
    }

    ...

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = new Intent(context, PollService.class);
        PendingIntent pi = PendingIntent.getService(
                context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
```

продолжение ⇨

Листинг 30.3 (*продолжение*)

```
                context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.RTC,
                    System.currentTimeMillis(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }

        PreferenceManager.getDefaultSharedPreferences(context)
            .edit()
            .putBoolean(PollService.PREF_IS_ALARM_ON, isOn)
            .commit();
    }
```

После этого `StartupReceiver` может использовать настройку для включения сигнала при загрузке.

Листинг 30.4. Включение сигнала при загрузке (StartupReceiver.java)

```
@Override
public void onReceive(Context context, Intent intent) {
    Log.i(TAG, "Received broadcast intent: " + intent.getAction());

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
    boolean isOn = prefs.getBoolean(PollService.PREF_IS_ALARM_ON, false);
    PollService.setServiceAlarm(context, isOn);
}
```

Снова запустите приложение PhotoGallery. На этот раз фоновый опрос должен автоматически перезапуститься после перезагрузки телефона, планшета или эмулятора.

Фильтрация оповещений переднего плана

Разобравшись с одним недостатком, мы обращаемся к другому изъяну в PhotoGallery. Оповещения прекрасно работают, но они отправляются даже тогда, когда приложение уже открыто.

Эта проблема также решается при помощи широковещательных интентов, но работать они будут совершенно иначе.

Отправка широковещательных интентов

Самая простая часть решения: отправка ваших собственных широковещательных интентов. Чтобы отправить широковещательный интент, просто создайте интент и передайте его `sendBroadcast(Intent)`. В нашем случае широковещательная рассылка будет применяться к определенному нами действию, поэтому также следует определить константу действия. Добавьте код из следующего листинга в `PollService`.

Листинг 30.5. Отправка широковещательного интента (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...

        if (!resultId.equals(lastResultId)) {
            ...

            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        }

        prefs.edit()
            .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
            .commit();
    }
```

Динамические широковещательные приемники

Следующий шаг — получение широковещательного интента. В принципе для этого можно написать широковещательный приемник, зарегистрированный в манифесте — наподобие `StartupReceiver`, но в нашем случае это не решит проблемы. Мы хотим, чтобы класс `PhotoGalleryFragment` получал интент только в то время, пока он живет. Автономный приемник, объявленный в манифесте, не сможет легко справиться с этой задачей. Он всегда будет получать интент и ему необходимо как-то узнать, что класс `PhotoGalleryFragment` живет.

Задача решается использованием *динамического широковещательного приемника*. Динамический приемник регистрируется в коде, а не в манифесте. Для регистрации приемника используется вызов `registerReceiver(BroadcastReceiver, IntentFilter)`, а для ее отмены — вызов `unregisterReceiver(BroadcastReceiver)`. Сам приемник обычно определяется как внутренний экземпляр, по аналогии со слушателем щелчка на кнопке. Но поскольку в `registerReceiver(…)` и `unregisterReceiver(…)` должен использоваться один экземпляр, приемник необходимо присвоить переменной экземпляра.

Создайте новый абстрактный класс `VisibleFragment`, суперклассом которого является `Fragment`. Этот класс будет представлять обобщенный фрагмент, скрывающий оповещения переднего плана (другой такой фрагмент мы напишем в главе 31).

Листинг 30.6. Класс VisibleFragment (VisibleFragment.java)

```
package com.bignerdranch.android.photogallery;
...

public abstract class VisibleFragment extends Fragment {
    public static final String TAG = "VisibleFragment";

    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                .show();
        }
    };

    @Override
    public void onResume() {
        super.onResume();
        IntentFilter filter = new
                              IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
    }

    @Override
    public void onPause() {
        super.onPause();
        getActivity().unregisterReceiver(mOnShowNotification);
    }
}
```

Обратите внимание: чтобы передать объект `IntentFilter`, необходимо создать его в коде. В данном случае объект `IntentFilter` идентичен фильтру, определяемому следующей разметкой XML:

```
<intent-filter>
    <action android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
</intent-filter>
```

Любой объект `IntentFilter`, который можно выразить в XML, также может быть представлен в коде подобным образом. Просто вызывайте `addCategory(String)`, `addAction(String)`, `addDataPath(String)` и так далее для настройки фильтра.

Динамически регистрируемые широковещательные приемники также должны принять меры для своей деинициализации. Как правило, если вы регистрируете приемник в методе жизненного цикла, вызываемом при запуске, в соответствующем методе завершения вызывается метод `Context.unregisterReceiver(BroadcastReceiver)`. В нашем примере регистрация выполняется в `onResume()` и отменяется

в onPause(). Аналогичным образом, если бы регистрация выполнялась в onActivityCreated(…), то отменяться она должна была бы в onActivityDestroyed().

(Кстати, будьте осторожны с onCreate(…) и onDestroy() при сохранении фрагментов. Метод getActivity() будет возвращать разные значения в onCreate(…) и onDestroy(), если экран был повернут. Если вы хотите регистрировать/отменять регистрацию в Fragment.onCreate(Bundle) и Fragment.onDestroy(), используйте getActivity().getApplicationContext()).

Сделайте PhotoGalleryFragment субклассом только что созданного класса VisibleFragment.

Листинг 30.7. Фрагмент делается видимым (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
public class PhotoGalleryFragment extends VisibleFragment {
    GridView mGridView;
    ArrayList<GalleryItem> mItems;
    ThumbnailDownloader<ImageView> mThumbnailThread;
```

Запустите PhotoGallery и пару раз переключите режим фонового опроса. Вы увидите, как наряду с бегущей строкой на экране появляется окно сообщения.

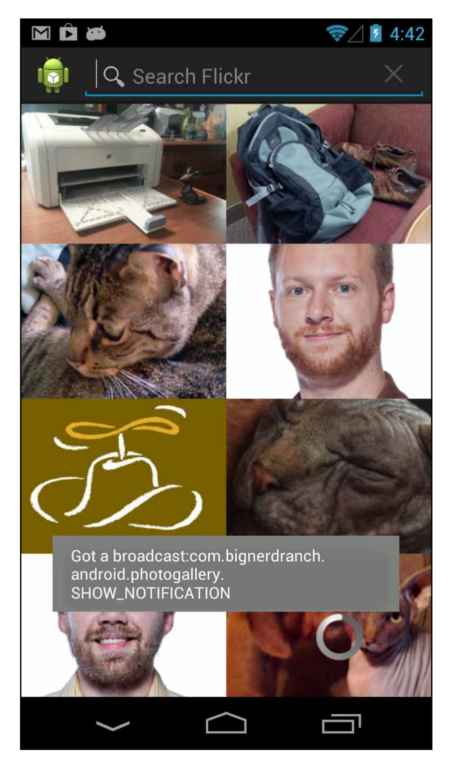


Рис. 30.3. Доказательство существования широковещательной рассылки

Закрытые разрешения

Одна из проблем с использованием широковещательной рассылки заключается в том, что любой компонент в системе может прослушивать ее или инициировать ваши приемники. И то и другое обычно нежелательно.

Эти несанкционированные вмешательства в ваши личные дела можно предотвратить парой способов. Если приемник объявлен в манифесте и является внутренним по отношению к вашему приложению, добавьте в тег `receiver` атрибут `android:exported="false"`. С ним приемник становится невидимым для других приложений в системе. В других ситуациях вы можете создать собственные разрешения, для чего в AndroidManifest.xml включается тег `permission`.

Добавьте следующий блок XML в AndroidManifest.xml, чтобы объявить и получить *закрытое* (private) разрешение.

Листинг 30.8. Добавление закрытого разрешения (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="17" />

  <permission android:name="com.bignerdranch.android.photogallery.PRIVATE"
    android:protectionLevel="signature" />

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
  <uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />

  <application
    ... >
    ...
  </application>

</manifest>
```

В этой разметке мы определяем собственное разрешение с *уровнем защиты* `signature` (вскоре об уровнях защиты будет рассказано более подробно). Само разрешение представляет собой простую строку — как и действия интентов, категории и системные разрешения, использовавшиеся ранее. Разрешение всегда необходимо получить для его использования, даже если вы сами определяете его. Таковы правила.

Обратите внимание на константу, выделенную цветом фона. Эта строка должна присутствовать в трех разных местах, и всюду она должна быть полностью идентичной. Лучше скопируйте ее вместо того, чтобы вводить вручную.

Чтобы использовать разрешение, определите соответствующую константу в коде и передайте ее при вызове `sendBroadcast(…)`.

Листинг 30.9. Отправка с разрешением (PollService.java)

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

    public static final String PERM_PRIVATE =
        "com.bignerdranch.android.photogallery.PRIVATE";

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...

        if (!resultId.equals(lastResultId)) {
            ...

            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        }

        prefs.edit()
            .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
            .commit();
    }
```

Чтобы использовать разрешение, передайте его в параметре `sendBroadcast(…)`. Теперь любое приложение сможет получить ваш интент, только указав то же разрешение, которое было указано при отправке.

Как насчет широковещательного приемника? Другая сторона сможет создать свой широковещательный интент, чтобы заставить ваш приемник сработать. Эта проблема также решается указанием разрешения при вызове `registerReceiver(…)`.

Листинг 30.10. Разрешения для широковещательного приемника (VisibleFragment.java)

```
@Override
    public void onResume() {
        super.onResume();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_
NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
        getActivity().registerReceiver(mOnShowNotification, filter,
            PollService.PERM_PRIVATE, null);
    }
```

Теперь только ваше приложение сможет заставить этот приемник сработать.

Подробнее об уровнях защиты

Каждое пользовательское разрешение должно задавать *уровень защиты* — атрибут `android:protectionLevel`. Уровень защиты сообщает Android, как будет использоваться разрешение. В нашем примере используется уровень защиты `signature`. Он означает, что если другое приложение захочет использовать ваше разрешение, то оно должно быть снабжено цифровой подписью с таким же ключом, как у вашего приложения. Обычно этот вариант оптимален для разрешений, используемых в вашем приложении. Так как ваш ключ недоступен для других разработчиков, они не могут получить доступ к функциональности, которую защищает ваше разрешение. Вдобавок, поскольку у вас есть собственный ключ, вы можете использовать разрешение в других приложениях, которые будут написаны позднее.

Таблица 30.1. Значения атрибута protectionLevel

Значение	Описание
normal	Используется для защиты функциональности приложения, которая не выполняет потенциально опасных операций — например, обращений к защищенным личным данным или отправки данных в Интернет. Пользователь видит разрешение перед установкой приложения, но не получает запроса на его явное предоставление. android.permission.RECEIVE_BOOT_COMPLETED использует этот уровень, как и разрешение на вибрацию телефона. Считайте, что речь идет о функциях, которые сами по себе не опасны, но вы хотите информировать пользователя о том, что они будут задействованы
dangerous	Используется для всего, для чего не используется normal — для обращений к личным данным, отправки и получению данных из сетевых интерфейсов, обращению к оборудованию, которое может использоваться для шпионских целей, и вообще всему, что может создать реальные проблемы для пользователя. В частности, разрешения на доступ к Интернету, камере и контактам относятся к этой категории. Android может запросить у пользователя подтверждение на выполнение опасной операции
signature	Система предоставляет это разрешение, если приложение подписано тем же сертификатом, что и приложение, в котором объявлено разрешение, или отклоняет его в противном случае. Если разрешение предоставляется, пользователь об этом не оповещается. Значение обычно используется для функциональности, внутренней для вашего приложения — так как у вас имеется сертификат, а приложение может использоваться только приложениями, подписанными тем же сертификатом, вы контролируете состав пользователей разрешения. В нашем случае значение не позволит посторонним видеть нашу широковещательную рассылку, но при желании вы можете написать другое приложение, которое также сможет ее прослушивать
signatureOrSystem	Аналог signature, но разрешение также предоставляется всем пакетам в образе системы Android. Используется для взаимодействия с приложениями, встроенными в образ системы, так что для вас, скорее всего, интереса не представляет

Получение результатов с упорядоченной широковещательной рассылкой

Мы организовали собственную широковещательную закрытую рассылку, но до сих пор передача данных осуществлялась только в одном направлении.

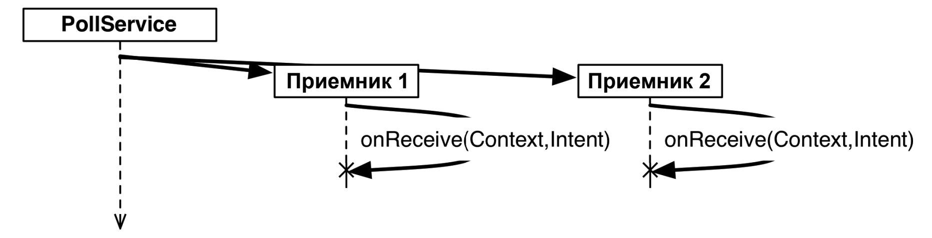


Рис. 30.4. Обычные широковещательные интенты

Это объяснялось тем, что концептуально обычный широковещательный интент принимается всеми одновременно. Сейчас `onReceive(…)` вызывается в главном потоке, поэтому на практике приемники не выполняются параллельно. Мы не можем рассчитывать на то, что они будут выполняться в каком-то конкретном порядке, или узнать, когда все они завершат выполнение. Этот факт значительно затрудняет взаимодействие между широковещательными приемниками или получение информации отправителем интента от приемников.

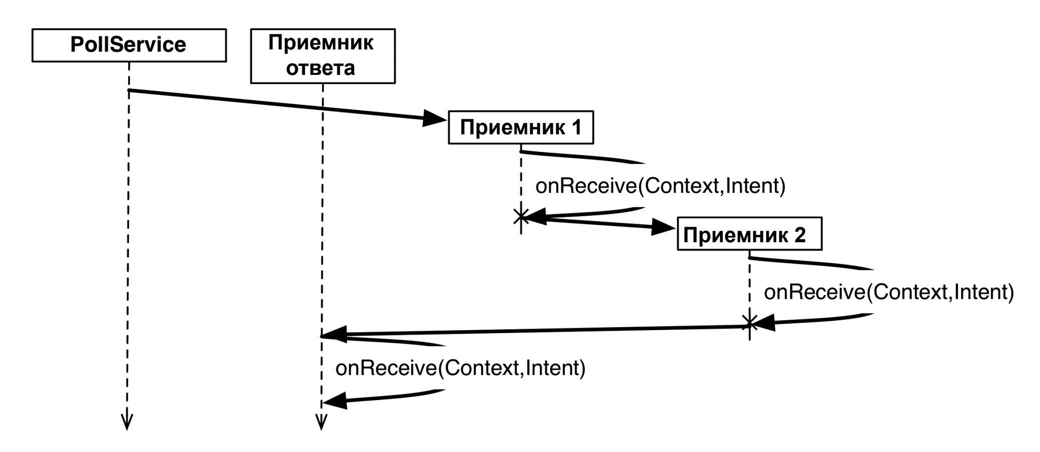


Рис. 30.5. Упорядоченные широковещательные интенты

Двустороннее взаимодействие можно реализовать с использованием *упорядоченных широковещательных интентов*. Упорядоченные широковещательные интенты позволяют серии широковещательных приемников обработать широковещательный интент по порядку. Кроме того, отправитель широковещательного интента может получать результаты, передавая при вызове специальный широковещательный приемник, называемый *получателем результата* (result receiver).

На получающей стороне все выглядит практически так же, как при обычной широковещательной рассылке. Однако в вашем распоряжении появляется дополнительный инструмент: набор методов, используемых для изменения возвращаемого значения вашего приемника. В нашей ситуации нужно отменить оповещение; эта информация передается в виде простого целочисленного кода результата. Соответственно мы используем метод `setResultCode(int)` для назначения кода результата `Activity.RESULT_CANCELED`.

Внесите изменения в `VisibleFragment`, чтобы вернуть информацию отправителю `SHOW_NOTIFICATION`.

Листинг 30.11. Возвращение простого результата (VisibleFragment.java)

```
private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(getActivity(),
                "Got a broadcast:" + intent.getAction(),
                Toast.LENGTH_LONG)
            .show();
        Log.i(TAG, "canceling notification");
        setResultCode(Activity.RESULT_CANCELED);
    }
};
```

Так как в нашем примере необходимо лишь подать сигнал «да/нет», нам достаточно кода результата. Если потребуется вернуть более сложные данные, используйте `setResultData(String)` или `setResultExtras(Bundle)`. А если вы захотите задать все три значения, вызовите `setResult(int,String,Bundle)`. После того как все возвращаемые значения будут заданы, каждый последующий приемник сможет увидеть или изменить их.

Чтобы эти методы делали что-то полезное, широковещательная передача должна быть упорядоченной. Напишите новый метод для отправки упорядоченных широковещательных интентов из `PollService`. Этот метод будет упаковывать обращение к `Notification` и отправлять его в широковещательном режиме. Получатель результата будет отправлять упакованный объект `Notification` (при условии, что он не был отменен).

Листинг 30.12. Отправка упорядоченных широковещательных интентов (PollService.java)

```
void showBackgroundNotification(int requestCode, Notification notification) {
    Intent i = new Intent(ACTION_SHOW_NOTIFICATION);
    i.putExtra("REQUEST_CODE", requestCode);
    i.putExtra("NOTIFICATION", notification);

    sendOrderedBroadcast(i, PERM_PRIVATE, null, null,
        Activity.RESULT_OK, null, null);
}
```

Метод `Context.sendOrderedBroadcast(Intent,String,BroadcastReceiver,Handler,int,String,Bundle)` имеет пять дополнительных параметров кроме используемых в `sendBroadcast(Intent,String)`. Вот они, по порядку: получатель результата, объект `Handler` для запуска получателя результата, а затем исходные значения

кода результата, данных результата и дополнения результата для упорядоченной широковещательной рассылки.

Получатель результата — специальный приемник, который выполняется после всех остальных приемников упорядоченного широковещательного интента. В других обстоятельствах мы смогли бы использовать получателя результата для получения широковещательного интента и отправки объекта оповещения. Однако в данном случае такое решение не сработает. Широковещательный интент часто будет отправляться непосредственно перед прекращением существования PollService. Это означает, что и приемник широковещательного интента может быть мертв.

Таким образом, последний приемник должен быть автономным. Создайте новый субкласс BroadcastReceiver с именем NotificationReceiver. Реализуйте его следующим образом.

Листинг 30.13. Реализация получателя результата (NotificationReceiver.java)

```
public class NotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "NotificationReceiver";

    @Override
    public void onReceive(Context c, Intent i) {
        Log.i(TAG, "received result: " + getResultCode());
        if (getResultCode() != Activity.RESULT_OK)
            // Активность переднего плана отменила
            // широковещательную передачу
            return;

        int requestCode = i.getIntExtra("REQUEST_CODE", 0);
        Notification notification = (Notification)
                i.getParcelableExtra("NOTIFICATION");

        NotificationManager notificationManager = (NotificationManager)
                c.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(requestCode, notification);
    }
}
```

Наконец, зарегистрируйте новый приемник. Поскольку он отправляет оповещение, получая набор результатов от других приемников, он должен выполняться после всех остальных. Это означает, что приемнику нужно назначить низкий приоритет. Так как он должен выполняться последним, назначьте ему приоритет –999 (значения –1000 и ниже зарезервированы).

А поскольку приемник используется только во внутренней работе приложения, его не обязательно делать видимым извне. Задайте атрибут android:exported="false", чтобы ограничить доступ к приемнику.

Листинг 30.14. Регистрация приемника оповещений (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  ... >
  ...

  <application
```

продолжение ⇗

Листинг 30.14 (*продолжение*)

```
    ... >
    ...
    <receiver android:name=".StartupReceiver">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
    <receiver android:name=".NotificationReceiver"
      android:exported="false">
      <intent-filter
        android:priority="-999">
        <action
            android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION"
 />
      </intent-filter>
    </receiver>
  </application>

</manifest>
```

Теперь используйте для отправки оповещения свой новый метод вместо `NotificationManager`.

Листинг 30.15. Последний шаг (PollService.java)

```
@Override
public void onHandleIntent(Intent intent) {
    ...

    if (!resultId.equals(lastResultId)) {
        ...
        Notification notification = new NotificationCompat.Builder(this)
            ...

            .build();

        NotificationManager notificationManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);

        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);

        showBackgroundNotification(0, notification);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}
```

Запустите приложение PhotoGallery и пару раз переключите режим фонового опроса. Вы увидите, что оповещения перестают появляться. Чтобы убедиться в том, что оповещения все еще работают в фоновом режиме, снова задайте значение

PollService.POLL_INTERVAL равным 5 секундам, чтобы вам не пришлось ждать целых пять минут.

Приемники и продолжительные задачи

Что же делать, если вы хотите, чтобы широковещательный интент запускал задачу более продолжительную, чем допускают ограничения главного цикла выполнения?

Есть два варианта. Первый — выделить эту работу в службу и запустить ее в широковещательном приемнике. Мы рекомендуем использовать именно этот способ. Служба может обрабатывать запрос столько времени, столько потребуется. Она может создать очередь из нескольких запросов и обслуживать их по порядку или обрабатывать запросы так, как считает нужным.

Второй вариант основан на использовании метода BroadcastReceiver.goAsync(). Этот метод возвращает объект BroadcastReceiver.PendingResult, который может использоваться для передачи результата в будущем. Таким образом, вы передаете объект PendingResult экземпляру AsyncTask для выполнения продолжительной работы, а потом отвечаете на широковещательную передачу, вызывая методы PendingResult.

У этого способа есть два недостатка. Во-первых, он недоступен на старых устройствах. Во-вторых, он менее гибок: вам все равно приходится обрабатывать широковещательную передачу за десять секунд или около того и в вашем распоряжении меньше архитектурных вариантов, чем при использовании службы.

Конечно, у метода goAsync() есть одно огромное преимущество: в нем можно задавать результаты упорядоченных широковещательных интентов. Если вам нужно именно это, другие решения не подойдут. Только позаботьтесь о том, чтобы выполнение не заняло слишком много времени.

31 Просмотр веб-страниц и WebView

С каждой фотографией, загружаемой с Flickr, связана страница. В этой главе мы сделаем так, чтобы пользователь мог нажать на фотографии в PhotoGallery и просмотреть ее страницу. Вы освоите два разных способа интеграции веб-контента в ваши приложения. Первый способ работает с браузером, установленным на устройстве, а второй использует класс `WebView` для отображения веб-контента.

И еще один блок данных Flickr

Для обоих способов нам понадобится URL-адрес страницы фотографии на Flickr. Если присмотреться к разметке XML, которую мы в настоящее время получаем для каждой фотографии, становится ясно, что страница в эти результаты не включена.

```
<photo id="8232706407" owner="70490293@N03" secret="9662732625"
    server="8343" farm="9" title="111_8Q1B2033" ispublic="1"
    isfriend="0" isfamily="0"
    url_s="http://farm9.staticflickr.com/8343/8232706407_9662732625_m.jpg"
    height_s="240" width_s="163" />
```

Похоже, придется писать новые запросы XML? К счастью, это не так. Обратившись к странице документации Flickr по адресу *http://www.flickr.com/services/api/misc.urls.html*, мы видим, что URL-адреса страниц отдельных фотографий строятся по схеме:

```
http://www.flickr.com/photos/идентификатор-пользователя/идентификатор-фото
```

Идентификатор фотографии совпадает со значением атрибута `id` в разметке XML. Мы уже сохранили его в поле `mId` объекта `GalleryItem`. Как насчет идентификатора пользователя? Немного покопавшись в документации, мы находим, что атрибут `owner` в XML содержит идентификатор пользователя. Таким образом, извлекая атрибут `owner`, мы можем построить URL-адрес по атрибутам из XML фотографии:

```
http://www.flickr.com/photos/owner/id
```

Чтобы реализовать этот план, включите следующий код в `GalleryItem`.

Листинг 31.1. Добавление кода построения URL страницы фотографии (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private String mOwner;

    ...

    public void setUrl(String url) {
        mUrl = url;
    }

    public String getOwner() {
        return mOwner;
    }

    public void setOwner(String owner) {
        mOwner = owner;
    }

    public String getPhotoPageUrl() {
        return "http://www.flickr.com/photos/" + mOwner + "/" + mId;
    }

    public String toString() {
        return mCaption;
    }
}
```

Здесь мы создаем новое свойство `mOwner` и добавляем короткий метод с именем `getPhotoPageUrl()` для построения URL-адреса страницы способом, описанным выше.

Теперь изменим метод `parseItems(…)` для чтения атрибута `owner`.

Листинг 31.2. Чтение атрибута owner (FlickrFetchr.java)

```
void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
        throws XmlPullParserException, IOException {
    int eventType = parser.next();

    while (eventType != XmlPullParser.END_DOCUMENT) {
        if (eventType == XmlPullParser.START_TAG &&
            XML_PHOTO.equals(parser.getName())) {
            String id = parser.getAttributeValue(null, "id");
            String caption = parser.getAttributeValue(null, "title");
            String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);
            String owner = parser.getAttributeValue(null, "owner");

            GalleryItem item = new GalleryItem();
            item.setUrl(smallUrl);
            item.setOwner(owner);
            items.add(item);
        }

        eventType = parser.next();
    }
}
```

Теперь можно поразвлечься с URL-адресом новой страницы.

Простой способ: неявные интенты

Сначала мы откроем страницу по этому URL-адресу при помощи старого знакомого — неявного интента. Этот интент запустит браузер с URL-адресом страницы фотографии.

Для начала нужно организовать прослушивание нажатий на элементах представления `GridView`. Это еще одно место, в котором наш код будет слегка отличаться от кода из главы 9 из-за отсутствия подходящей реализации `GridFragment`. Вместо переопределения метода `onListItemClick(…)` в фрагменте мы подключим обработчик в стиле слушателя щелчков на кнопке, вызывая `setOnItemClickListener(…)` в `GridView`.

После этого остается лишь создать и отправить неявный интент. Добавьте следующий код в `PhotoGalleryFragment`.

Листинг 31.3. Просмотр веб-страниц с использованием неявных интентов (PhotoGalleryFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

    mGridView = (GridView)v.findViewById(R.id.gridView);

    setupAdapter();

    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
                long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);

            startActivity(i);
        }
    });

    return v;
}
```

Запустите приложение PhotoGallery и нажмите на фотографии. На экране ненадолго появляется индикатор прогресса, после чего открывается браузер.

Более сложный способ: WebView

На практике веб-контент чаще требуется отобразить в активности вашего приложения вместо перехода в браузер. Допустим, вы хотите отобразить самостоятельно сгенерированную разметку HTML или просто обойтись без браузера. Справочная документация в приложениях часто реализуется в виде веб-страницы, чтобы ее было

удобнее обновлять. Запуск браузера для просмотра справочных страниц выглядит непрофессионально и препятствует изменению поведения, а также интеграции веб-страницы в ваш пользовательский интерфейс.

Для представления веб-контента в пользовательском интерфейсе приложения используется класс WebView. Мы назвали этот способ «более сложным», но на самом деле он очень прост (хотя по сравнению с неявными интентами все можно назвать сложным).

Нашим первым шагом станет создание новой активности и фрагмента для отображения WebView. Начнем, как обычно, с определения файла макета.

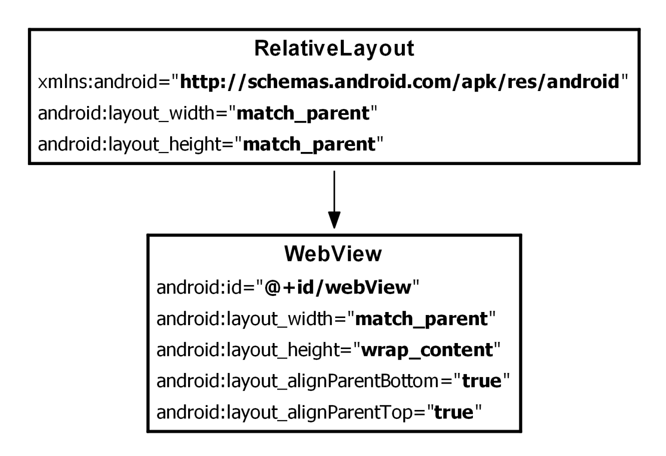


Рис. 31.1. Исходный макет (res/layout/fragment_photo_page.xml)

При взгляде на диаграмму возникает мысль: «От RelativeLayout нет никакой пользы». И верно — однако позднее в этой главе мы наполним его дополнительным «хромом».

Создайте PhotoPageFragment как субкласс класса VisibleFragment, созданного в предыдущей главе. Необходимо заполнить файл макета, выделить из него WebView и передать URL-адрес в данных интента.

Листинг 31.4. Создание фрагмента браузера (PhotoPageFragment.java)

```
package com.bignerdranch.android.photogallery;
...

public class PhotoPageFragment extends VisibleFragment {
    private String mUrl;
    private WebView mWebView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mUrl = getActivity().getIntent().getData().toString();
    }
```

продолжение ⇨

Листинг 31.4 (*продолжение*)

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

        mWebView = (WebView)v.findViewById(R.id.webView);

        return v;
    }
}
```

Пока это всего лишь заготовка — вскоре мы заполним ее кодом. А пока создайте класс-контейнер PhotoPageActivity на основе хорошо знакомого класса SingleFragmentActivity.

Листинг 31.5. Создание веб-активности (PhotoPageActivity.java)

```
package com.bignerdranch.android.photogallery;
...

public class PhotoPageActivity extends SingleFragmentActivity {
    @Override
    public Fragment createFragment() {
        return new PhotoPageFragment();
    }
}
```

Измените код PhotoGalleryFragment, чтобы вместо неявного интента происходило обращение к новой активности.

Листинг 31.6. Переключение на обращение к новой активности (PhotoGalleryFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...

    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
                long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);
            Intent i = new Intent(getActivity(), PhotoPageActivity.class);
            i.setData(photoPageUri);

            startActivity(i);
        }
    });

    return v;
}
```

Наконец, добавьте новую активность в манифест.

Листинг 31.7. Добавление активности в манифест (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  ...

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".PhotoGalleryActivity"
      android:launchMode="singleTop"
      android:label="@string/title_activity_photo_gallery" >
      ...
    </activity>
    <activity
        android:name=".PhotoPageActivity" />
    <service android:name=".PollService" />
    <receiver android:name=".StartupReceiver">
        ...
    </receiver>
  </application>
</manifest>
```

Запустите приложение PhotoGallery и нажмите на фотографии. На экране появляется новая пустая активность.

Возьмемся за дело и заставим наш фрагмент делать что-то полезное. Чтобы виджет `WebView` успешно отображал страницу фотографии на сайте Flickr, необходимо выполнить три условия. Первое условие очевидно — нужно сообщить ему, какой URL-адрес необходимо загрузить.

Второе условие — необходимо включить поддержку JavaScript. По умолчанию она отключена. Постоянно держать ее включенной не обязательно, но для Flickr она нужна. Android Lint выдает предупреждение (из-за потенциальной опасности межсайтовых сценарных атак), так что предупреждения Lint тоже нужно отключить.

Наконец, необходимо переопределить в классе `WebViewClient` один метод с именем `shouldOverrideUrlLoading(WebView,String)` и вернуть `false`. Мы рассмотрим этот класс после того, как вы введете код.

Листинг 31.8. Добавление новых переменных экземпляров (PhotoPageFragment.java)

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    mWebView = (WebView)v.findViewById(R.id.webView);
```

продолжение ⇗

Листинг 31.8 (*продолжение*)

```
    mWebView.getSettings().setJavaScriptEnabled(true);

    mWebView.setWebViewClient(new WebViewClient() {
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            return false;
        }
    });

    mWebView.loadUrl(mUrl);

    return v;
}
```

Загрузка данных с URL-адреса должна происходить после настройки `WebView`, поэтому она выполняется в последнюю очередь. До этого мы включаем JavaScript, вызывая `getSettings()` для получения экземпляра `WebSettings`, с последующим вызовом `WebSettings.setJavaScriptEnabled(true)`. Объект `WebSettings` — первый из трех механизмов настройки `WebView`. Он содержит различные свойства, которые можно задать в коде — например, строку пользовательского агента и размер текста.

Далее выполняется настройка `WebViewClient`. `WebViewClient` представляет собой событийный интерфейс. Предоставляя собственную реализацию `WebViewClient`, вы можете реагировать на события вывода. Например, можно определить, когда ядро визуализации начинает загрузку изображения с конкретного URL-адреса, или решить, стоит ли заново отправить серверу POST-запрос.

`WebViewClient` содержит много методов, которые можно переопределить; большинство этих методов нам не понадобится. Однако мы должны заменить стандартную реализацию `shouldOverrideUrlLoading(WebView,String)` из `WebViewClient`. Этот метод указывает, что должно происходить при загрузке нового URL-адреса в `WebView` (например, при нажатии на ссылке). Если он возвращает `true`, это означает: «Не обрабатывать этот URL-адрес, я обрабатываю его сам». Если он возвращает `false`, вы говорите: «Давай, `WebView`, загружай данные с URL-адреса, я с ним ничего не делаю».

Реализация по умолчанию инициирует неявный интент с URL, по аналогии с тем, как это делалось ранее в этой главе. Для страницы с фотографией это создает серьезные проблемы, потому что Flickr первым делом выполняет перенаправление на мобильную версию сайта. Для `WebViewClient` это означает немедленное переключение на браузер по умолчанию; совсем не то, что нам нужно.

Проблема решается просто — переопределите реализацию по умолчанию, чтобы она возвращала `false`.

Запустите приложение PhotoGallery, и вы увидите `WebView` на экране.

Класс WebChromeClient

Раз уж мы занялись созданием собственной реализации `WebView`, давайте немного украсим ее, добавив представление заголовка и индикатор прогресса. Откройте файл fragment_photo_page.xml и внесите следующие изменения.

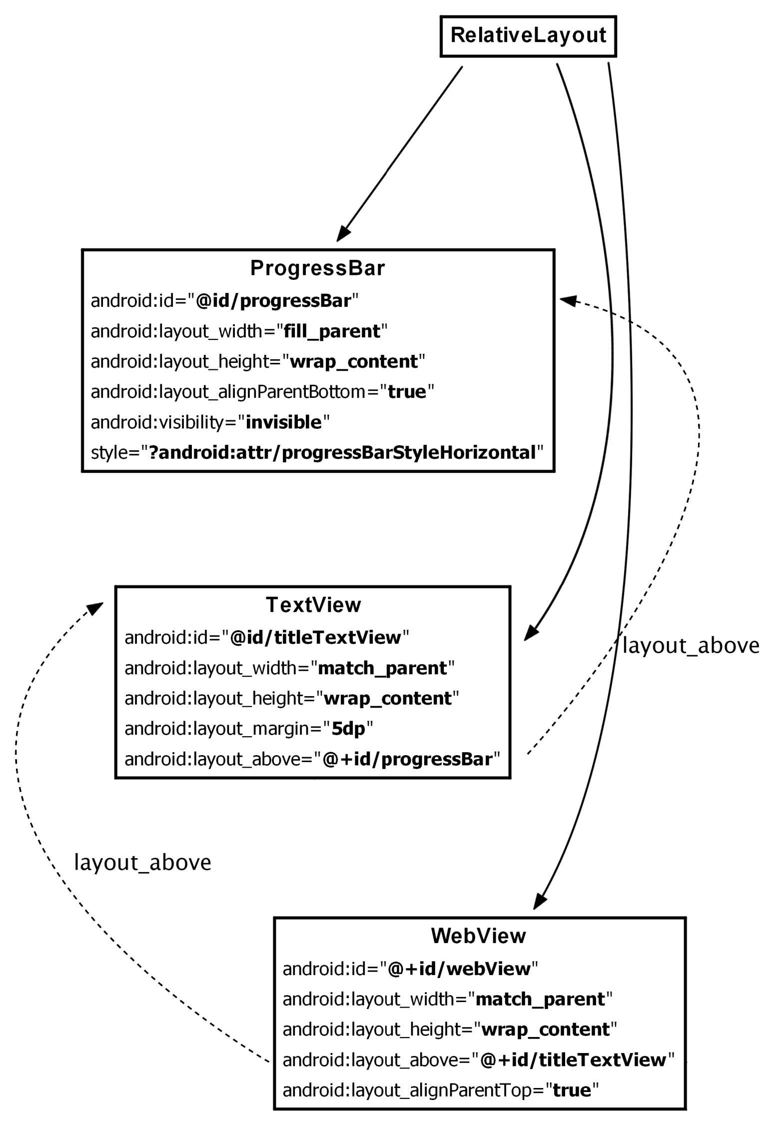


Рис. 31.2. Добавление заголовка и индикатора прогресса (fragment_photo_page.xml)

Добавить `ProgressBar` и `TextView` несложно, но чтобы подключить их, нам понадобится вторая точка обратного вызова `WebView`: `WebChromeClient`. Если `WebViewClient` определяет интерфейс обработки событий визуализации, `WebChromeClient` определяет событийный интерфейс обработки событий, которые должны изменять элементы «хрома» (chrome) в браузере. К этой категории относятся сигналы (alerts) JavaScript, значки сайтов favicon, обновления прогресса загрузки и т. д.

Подключите его в методе `onCreateView(…)`.

Листинг 31.9. Использование WebChromeClient (PhotoPageFragment.java)

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    final ProgressBar progressBar = (ProgressBar)v.findViewById(R.id.progressBar);
    progressBar.setMax(100);  // значения в диапазоне 0-100
    final TextView titleTextView = (TextView)v.findViewById(R.id.titleTextView);

    mWebView = (WebView)v.findViewById(R.id.webView);

    mWebView.getSettings().setJavaScriptEnabled(true);

    mWebView.setWebViewClient(new WebViewClient() {
        ...
    });

    mWebView.setWebChromeClient(new WebChromeClient() {
        public void onProgressChanged(WebView webView, int progress) {
            if (progress == 100) {
                progressBar.setVisibility(View.INVISIBLE);
            } else {
                progressBar.setVisibility(View.VISIBLE);
                progressBar.setProgress(progress);
            }
        }

        public void onReceivedTitle(WebView webView, String title) {
            titleTextView.setText(title);
        }
    });

    mWebView.loadUrl(mUrl);

    return v;
}
```

Обновления индикатора прогресса и заголовка имеют собственные методы обратного вызова, onProgressChanged(WebView,int) и onReceivedTitle(WebView, String). Информация о прогрессе, получаемая от onProgressChanged(WebView,int), представляет собой целое число от 0 до 100. Если результат равен 100, значит, загрузка страницы завершена, поэтому мы скрываем ProgressBar, задавая режим View.INVISIBLE.

Запустите приложение PhotoGallery и протестируйте внесенные изменения.

Повороты в WebView

Попробуйте повернуть экран. Хотя приложение работает правильно, вы заметите, что WebView полностью перезагружает веб-страницу. Дело в том, что у WebView слишком много данных, чтобы сохранить их все в onSaveInstanceState(…), и их

приходится создавать «с нуля» каждый раз, когда их приходится создавать заново при повороте.

Для таких классов (другой пример — VideoView) документация Android рекомендует позволить активности самой обработать все изменения конфигурации. Это означает, что вместо уничтожения активности она просто перемещает свои представления для размещения по новым размерам экрана. В результате WebView не приходится заново загружать свои данные.

(Почему бы не делать так всегда, спросите вы? Такое решение не всегда работает правильно с любыми представлениями. Иначе жизнь разработчика была бы намного проще, но... увы.)

Чтобы заставить класс PhotoPageActivity обрабатывать свои изменения конфигурации, внесите следующее изменения в AndroidManifest.xml.

Листинг 31.10. Самостоятельный обработчик изменений конфигурации (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >
  ...

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
    <activity
        android:name=".PhotoPageActivity"
        android:configChanges="keyboardHidden|orientation|screenSize" />
    ...
  </application>

</manifest>
```

Атрибут сообщает, что в случае изменения конфигурации из-за открытия или закрытия клавиатуры, изменения ориентации или размеров экрана (которое также происходит при переключении между книжной и альбомной ориентацией в Android после версии 3.2) активность должна обрабатывать изменения самостоятельно.

Вот и все. Попробуйте снова повернуть устройство; на этот раз все должно быть в ажуре.

Для любознательных: внедрение объектов JavaScript

Вы уже видели, как следует использовать WebViewClient и WebChromeClient для обработки некоторых событий, происходящих в WebView. Однако еще больше возможностей открывает внедрение произвольных объектов JavaScript в документ,

содержащийся в виджете `WebView`. Откройте документацию по адресу *http://developer.android.com/reference/android/webkit/WebView.html* и прокрутите до описания метода `addJavascriptInterface(Object,String)`. Этот метод позволяет внедрить произвольный объект в документ с заданным именем.

```
mWebView.addJavascriptInterface(new Object() {
    public void send(String message) {
        Log.i(TAG, "Received message: " + message);
    }
}, "androidObject");
```

После этого объект используется следующим образом:

```
<input type="button" value="In WebView!"
    onClick="sendToAndroid('In Android land')" />

<script type="text/javascript">
    function sendToAndroid(message) {
        androidObject.send(message);
    }
</script>
```

Такое решение весьма рискованно — фактически вы разрешаете потенциально небезопасной веб-странице вмешиваться в работу вашей программы. Следовательно, его стоит применять только для принадлежащей вам разметки HTML — или ограничиться предоставлением в высшей степени консервативного интерфейса.

32 Пользовательские представления и события касания

В этой главе мы займемся обработкой событий касания. Для этого мы создадим субкласс `View` с именем `BoxDrawingView`. На этом представлении пользователь рисует прямоугольники, прикасаясь к экрану и перемещая палец.

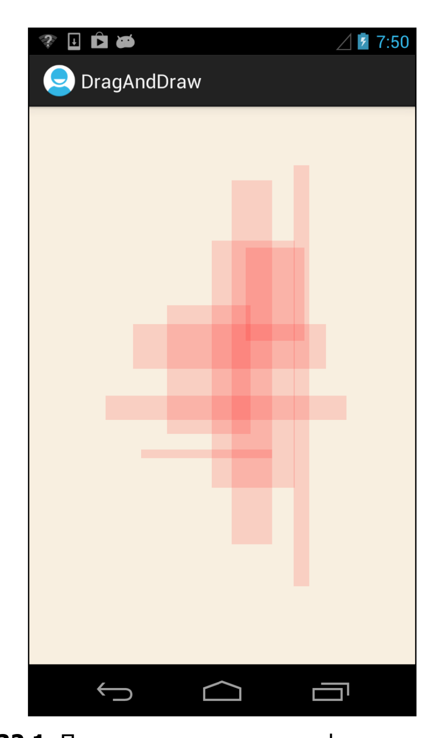


Рис. 32.1. Прямоугольники разных форм и размеров

Создание проекта DragAndDraw

Класс BoxDrawingView занимает центральное место в новом проекте DragAndDraw. Выполните команду New ▸ Android Application Project. Задайте параметры проекта, как показано на рис. 32.2, и создайте пустую активность с именем DragAndDrawActivity.

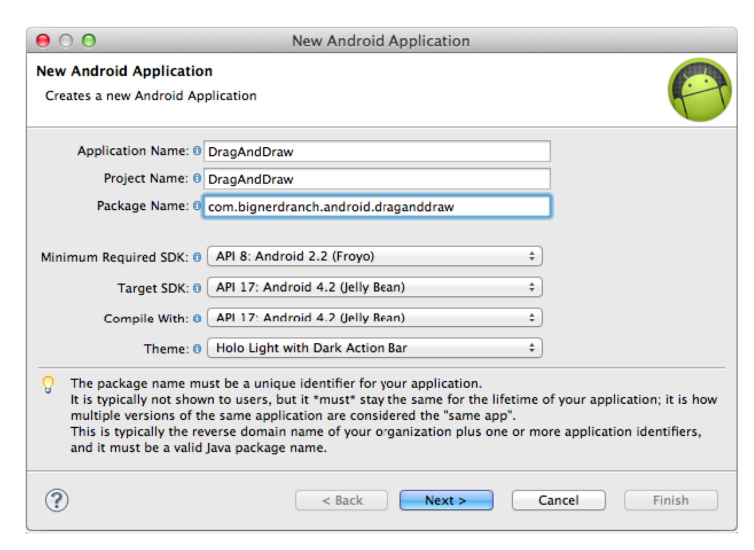


Рис. 32.2. Создание проекта DragAndDraw

Создание класса DragAndDrawActivity

Класс DragAndDrawActivity представляет собой субкласс SingleFragmentActivity, который заполняет обычный макет с одним фрагментом. На панели Package Explorer скопируйте файл SingleFragmentActivity.java в пакет com.bignerdranch.android.draganddraw. Затем скопируйте файл activity_fragment.xml в каталог res/layout проекта DragAndDraw.

В файле DragAndDrawActivity.java объявите DragAndDrawActivity субклассом SingleFragmentActivity. Этот класс должен создавать экземпляр DragAndDrawFragment (класс, который будет создан следующим).

Листинг 32.1. Изменение активности (DragAndDrawActivity.java)

```
public class DragAndDrawActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drag_and_draw);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_drag_and_draw, menu);
        return true;
    }
```

```
    @Override
    public Fragment createFragment() {
        return new DragAndDrawFragment();
    }
}
```

Создание класса DragAndDrawFragment

Чтобы подготовить макет DragAndDrawFragment, переименуйте файл activity_drag_and_draw.xml в fragment_drag_and_draw.xml.

Макет DragAndDrawFragment в конечном итоге будет состоять из BoxDrawingView — пользовательского представления, которое мы собираемся написать. Весь графический вывод и обработка событий касания будут реализованы в BoxDrawingView.

Создайте новый класс с именем DragAndDrawFragment и назначьте его суперклассом android.support.v4.app.ListFragment. Переопределите метод onCreateView(…), чтобы он заполнял макет fragment_drag_and_draw.xml.

Листинг 32.2. Создание фрагмента (DragAndDrawFragment.java)

```
public class DragAndDrawFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_drag_and_draw, parent, false);
        return v;
    }
}
```

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно.

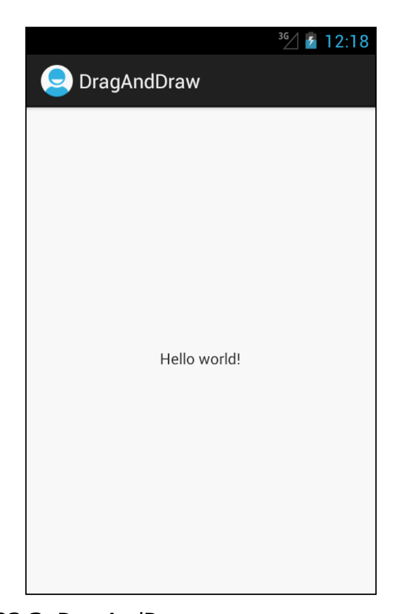


Рис. 32.3. DragAndDraw с макетом по умолчанию

Создание нестандартного представления

Android предоставляет много превосходных стандартных представлений и виджетов, но иногда требуется создать нестандартное представление с визуальным оформлением, полностью уникальным для вашего приложения.

Все многообразие нестандартных представлений можно условно разделить на две общие категории:

- *простые представления* — простое представление может быть устроено достаточно сложно; «простым» оно называется только потому, что не имеет дочерних представлений. Простое представление почти всегда также выполняет нестандартную прорисовку;
- *составные представления* — состоят из других объектов представлений. Составные представления обычно управляют дочерними представлениями, но не занимаются своей прорисовкой. Вместо этого каждому дочернему представлению делегируется своя часть работы по прорисовке.

Создание нестандартного представления состоит из трех шагов:

- Выбор суперкласса. Для простого нестандартного представления `View` предоставляет пустой «холст» для рисования, поэтому этот выбор является наиболее распространенным. Для составных нестандартных представлений выберите подходящий класс макета.
- Субклассируйте выбранный класс и переопределите как минимум один конструктор из суперкласса или создайте собственный конструктор, вызывающий один из конструкторов суперкласса.
- Переопределите другие ключевые методы для настройки поведения.

Создание класса BoxDrawingView

Класс `BoxDrawingView` относится к категории простых представлений и является прямым субклассом `View`.

Создайте новый класс с именем `BoxDrawingView` и назначьте `View` его суперклассом. Добавьте в файл BoxDrawingView.java два конструктора.

Листинг 32.3. Исходная реализация BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    // Используется при создании представления в коде
    public BoxDrawingView(Context context) {
        this(context, null);
    }

    // Используется при заполнении представления по разметке XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}
```

Два конструктора нужны потому, что экземпляр представления может создаваться как в коде, так и по файлу разметки. Представления, созданные на базе файла макета,

получают экземпляр AttributeSet с атрибутами XML, заданными в XML. Даже если вы не собираетесь использовать оба конструктора, их рекомендуется включить.

Затем обновите файл макета fragment_drag_and_draw.xml, чтобы в нем использовалось новое представление.

Листинг 32.4. Включение Add BoxDrawingView в макет (fragment_drag_and_draw.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
 >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />

</RelativeLayout>

<com.bignerdranch.android.draganddraw.BoxDrawingView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

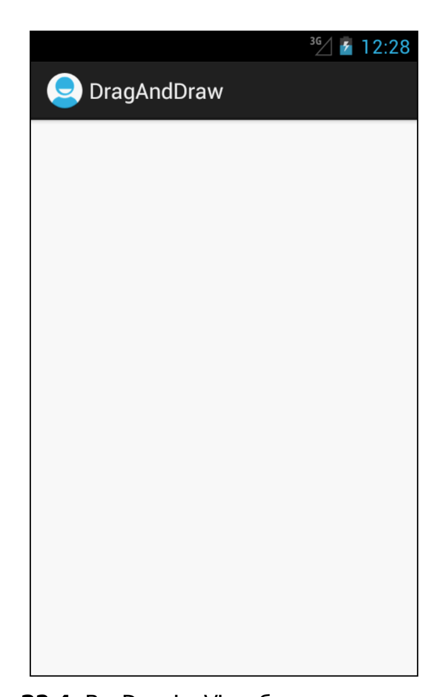


Рис. 32.4. BoxDrawingView без прямоугольников

Чтобы заполнитель макетов нашел класс BoxDrawingView, вы должны использовать полностью уточненное имя. Заполнитель просматривает файл макета, создавая экземпляры View. Если имя элемента будет неполным, то заполнитель ищет класс с указанным именем в пакетах android.view и android.widget. Если класс находится в другом месте, заполнитель его не найдет, и в приложении произойдет сбой. По этой причине для классов, не входящих в android.view и android.widget, необходимо всегда задавать полностью уточненное имя.

Запустите приложение DragAndDraw и убедитесь в том, что настройка была выполнена правильно. Правда, пока на экране нет ничего, кроме пустого представления.

На следующем этапе мы научим BoxDrawingView прослушивать события касания и использовать содержащуюся в них информацию для рисования прямоугольников на экране.

Обработка событий касания

Для прослушивания событий касания можно назначить слушателя события при помощи следующего метода класса View:

```
public void setOnTouchListener(View.OnTouchListener l)
```

Этот метод работает почти так же, как setOnClickListener(View.OnClickListener). Вы предоставляете реализацию View.OnTouchListener, а слушатель вызывается каждый раз, когда происходит событие касания.

Но поскольку мы субклассируем View, можно пойти по сокращенному пути и переопределить следующий метод класса View:

```
public boolean onTouchEvent(MotionEvent event)
```

Этот метод получает экземпляр MotionEvent — класса, описывающего событие касания, включая его позицию и действие. Действие описывает стадию события.

Константы действий	Описание
ACTION_DOWN	Пользователь прикоснулся к экрану
ACTION_MOVE	Пользователь перемещает палец по экрану
ACTION_UP	Пользователь отводит палец от экрана
ACTION_CANCEL	Родительское представление перехватило событие касания

В своей реализацияи onTouchEvent(…) для проверки действия можно воспользоваться следующим методом класса MotionEvent:

```
public final int getAction()
```

Добавьте в файл BoxDrawingView.java тег для журнала и реализацию onTouch(…), которая регистрирует в журнале информацию о каждом из четырех разных действий.

Листинг 32.5. Реализация BoxDrawingView (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";
    ...

    public boolean onTouchEvent(MotionEvent event) {
        PointF curr = new PointF(event.getX(), event.getY());

        Log.i(TAG, "Received event at x=" + curr.x +
                ", y=" + curr.y + ":");
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                Log.i(TAG, "  ACTION_DOWN");
                break;
            case MotionEvent.ACTION_MOVE:
                Log.i(TAG, "  ACTION_MOVE");
                break;
            case MotionEvent.ACTION_UP:
                Log.i(TAG, "  ACTION_UP");
                break;
            case MotionEvent.ACTION_CANCEL:
                Log.i(TAG, "  ACTION_CANCEL");
                break;
        }

        return true;
    }
}
```

Обратите внимание: координаты X и Y упаковываются в объекте `PointF`. В оставшейся части этой главы эти два значения обычно будут передаваться вместе. `PointF` — предоставленный Android класс-контейнер, который решает эту задачу за вас.

Запустите приложение DragAndDraw и откройте LogCat. Прикоснитесь к экрану и проведите пальцем. Вы увидите в журнале сообщения с координатами X и Y каждого действия касания, полученного `BoxDrawingView`.

Отслеживание перемещений между событиями

Класс `BoxDrawingView` должен рисовать прямоугольники, а не регистрировать координаты. Для этого необходимо решить ряд задач.

Прежде всего для определения прямоугольника нам понадобятся:

- базовая точка (в которой было сделано исходное касание);
- текущая точка (в которой находится палец).

Следовательно, для определения прямоугольника необходимо отслеживать данные от нескольких событий `MotionEvent`. Данные будут храниться в объекте `Box`.

Создайте класс с именем `Box` для хранения данных, определяющих прямоугольник.

Листинг 32.6. Класс Box (Box.java)

```
public class Box {
    private PointF mOrigin;
    private PointF mCurrent;

    public Box(PointF origin) {
        mOrigin = mCurrent = origin;
    }

    public void setCurrent(PointF current) {
        mCurrent = current;
    }

    public PointF getOrigin() {
        return mOrigin;
    }
}
```

Когда пользователь прикасается к `BoxDrawingView`, новый объект `Box` создается и включается в массив существующих прямоугольников (рис. 32.5).

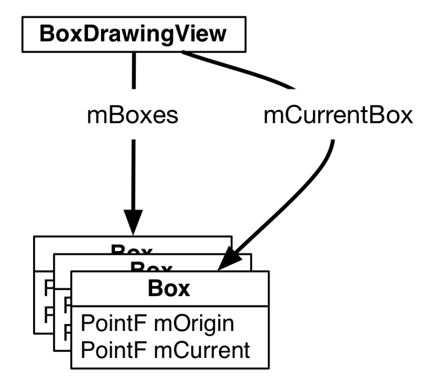


Рис. 32.5. Объекты в DragAndDraw

Добавьте в `BoxDrawingView` код, использующий новый объект `Box` для отслеживания текущего состояния рисования.

Листинг 32.7. Добавление методов жизненного цикла событий касания (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private ArrayList<Box> mBoxes = new ArrayList<Box>();

    ...

    public boolean onTouchEvent(MotionEvent event) {
        PointF curr = new PointF(event.getX(), event.getY());
```

```
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                Log.i(TAG, "  ACTION_DOWN");
                // Reset drawing state
                mCurrentBox = new Box(curr);
                mBoxes.add(mCurrentBox);
                break;

            case MotionEvent.ACTION_MOVE:
                Log.i(TAG, "  ACTION_MOVE");
                if (mCurrentBox != null) {
                    mCurrentBox.setCurrent(curr);
                    invalidate();
                }
                break;

            case MotionEvent.ACTION_UP:
                Log.i(TAG, "  ACTION_UP");
                mCurrentBox = null;
                break;

            case MotionEvent.ACTION_CANCEL:
                Log.i(TAG, "  ACTION_CANCEL");
                mCurrentBox = null;
                break;
        }

        return true;
    }
}
```

При каждом получении события ACTION_DOWN в поле mCurrentBox сохраняется новый объект Box с базовой точкой, соответствующей позиции события. Этот объект Box добавляется в массив прямоугольников (в следующем разделе, когда мы займемся прорисовкой, BoxDrawingView будет выводить каждый объект Box из массива).

В процессе перемещения пальца по экрану приложение обновляет mCurrentBox.mCurrent. Затем, когда касание отменяется или палец отходит от экрана, поле mCurrentBox обнуляется для завершения операции. Объект Box завершен; он сохранен в массиве и уже не будет обновляться событиями перемещения.

Обратите внимание на вызов invalidate() в случае ACTION_MOVE. Он заставляет BoxDrawingView перерисовать себя, чтобы пользователь видел прямоугольник в процессе перетаскивания. Мы подошли к следующему шагу: рисованию прямоугольников на экране.

Рисование внутри onDraw(…)

При запуске приложения все его представления недействительны (invalid). Это означает, что они ничего не вывели на экран. Для исправления ситуации Android вызывает метод draw() объекта View верхнего уровня. В результате представление перерисовывает себя, что заставляет его потомков перерисовать себя. Затем потомки этих потомков перерисовывают себя и так далее вниз по иерархии. Когда все

представления в иерархии перерисуют себя, объект View верхнего уровня перестает быть недействительным.

Чтобы вмешаться в процесс прорисовки, следует переопределить следующий метод View:

```
protected void onDraw(Canvas canvas)
```

Вызов invalidate(), выполняемый в ответ на действие ACTION_MOVE в onTouchEvent(…), снова делает объект BoxDrawingView недействительным. Это заставляет его перерисовать себя и приводит к повторному вызову onDraw(…).

Обратите внимание на параметр Canvas. Canvas и Paint — два главных класса, используемых при рисовании в Android.

Класс Canvas содержит все выполняемые операции графического вывода. Методы, вызываемые для объекта Canvas, определяют, где и что выводится — линия, круг, слово или прямоугольник.

Класс Paint определяет, как будут выполняться эти операции. Методы, вызываемые для объекта Paint, определяют характеристики вывода — должны ли фигуры заполняться, каким шрифтом должен выводиться текст, каким цветом должны выводиться линии и т. д.

В файле BoxDrawingView.java создайте два объекта Paint в конструкторе BoxDrawingView для XML.

Листинг 32.8. Создание объектов Paint (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private ArrayList<Box> mBoxen = new ArrayList<Box>();
    private Box mCurrentBox;
    private Paint mBoxPaint;
    private Paint mBackgroundPaint;

    ...

    // Используется при заполнении представления по разметке XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);

        // Прямоугольники рисуются полупрозрачным красным цветом (ARGB)
        mBoxPaint = new Paint();
        mBoxPaint.setColor(0x22ff0000);

        // Фон закрашивается серовато-белым цветом
        mBackgroundPaint = new Paint();
        mBackgroundPaint.setColor(0xfff8efe0);
    }
}
```

После создания объектов Paint можно переходить к рисованию прямоугольников на экране.

Листинг 32.9. Переопределение onDraw(Canvas) (BoxDrawingView.java)

```
@Override
protected void onDraw(Canvas canvas) {
    // Заполнение фона
    canvas.drawPaint(mBackgroundPaint);

    for (Box box : mBoxes) {
        float left = Math.min(box.getOrigin().x, box.getCurrent().x);
        float right = Math.max(box.getOrigin().x, box.getCurrent().x);
        float top = Math.min(box.getOrigin().y, box.getCurrent().y);
        float bottom = Math.max(box.getOrigin().y, box.getCurrent().y);

        canvas.drawRect(left, top, right, bottom, mBoxPaint);
    }
}
```

Первая часть кода тривиальна: используя серовато-белый цвет, мы заполняем «холст» задним фоном для вывода прямоугольников.

Затем для каждого прямоугольника в списке мы определяем значения `left`, `right`, `top` и `bottom` по двум точкам. Значения `left` и `top` будут минимальными, а `bottom` и `right` — максимальными.

После вычисления параметров вызов метода `Canvas.drawRect(…)` рисует красный прямоугольник на экране.

Запустите приложение DragAndDraw и нарисуйте несколько прямоугольников.

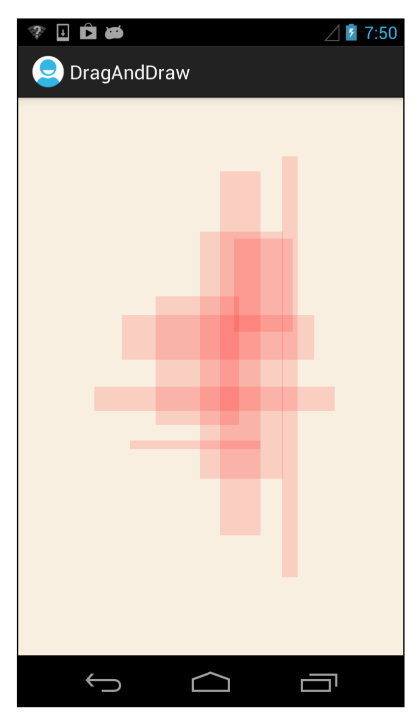


Рис. 32.6. Приложение с нарисованными прямоугольниками

Упражнение. Повороты

Как сохранить прямоугольники из `View` при изменении ориентации? Например, это можно сделать при помощи следующих методов `View`:

```
protected Parcelable onSaveInstanceState()
protected void onRestoreInstanceState(Parcelable state)
```

Эти методы работают не так, как метод `onSaveInstanceState(Bundle)` классов `Activity` и `Fragment`. Вместо объекта `Bundle` они возвращают и обрабатывают объект, реализующий интерфейс `Parcelable`. Мы рекомендуем использовать `Bundle` в качестве `Parcelable` вместо того, чтобы писать реализацию `Parcelable` самостоятельно. (Реализация интерфейса `Parcelable` весьма сложна. Лучше избегать ее там, где это возможно.)

Еще одно, более сложное упражнение: реализуйте возможность вращения прямоугольников вторым пальцем. Для этого вам потребуется отслеживать операции с несколькими указателями в коде обработки `MotionEvent`.

При работе с множественными касаниями вам понадобятся:

- *индекс указателя* — сообщает, к какому указателю в текущем наборе относится событие;
- *идентификатор указателя* — обеспечивает однозначную идентификацию конкретного пальца в жесте.

Индекс указателя может изменяться, но идентификатор остается неизменным.

За дополнительной информацией обращайтесь к документации по следующим методам `MotionEvent`:

```
public final int getActionMasked()
public final int getActionIndex()
public final int getPointerId(int pointerIndex)
public final float getX(int pointerIndex)
public final float getY(int pointerIndex)
```

Также посмотрите документацию по константам `ACTION_POINTER_UP` и `ACTION_POINTER_DOWN`.

33 Отслеживание местоположения устройства

В этой главе мы создадим новое приложение RunTracker, работающее с подсистемой GPS устройства для записи и отображения перемещений пользователя: прогулок по лесу, поездок на машине и океанских круизов. Приложение RunTracker будет помнить все.

Первая версия RunTracker просто получает обновленную информацию местоположения от GPS и отображает текущую позицию устройства на экране. В итоговом варианте RunTracker будет выводить карту, отслеживающую перемещения пользователя в реальном времени.

Создание приложения RunTracker

Создайте новое приложение Android со следующей конфигурацией, показанной на рис. 33.1.

Обратите внимание на два отличия от предыдущих проектов. Во-первых, минимальная версия SDK повышается до API 9. Во-вторых, построение производится для последней версии Google API, а не для версии Android. Google API понадобится нам для работы с картами.

Если вы не видите строку Google APIs в списке целей, загрузите ее из Android SDK Manager. Выполните команду Window ▸ Android SDK Manager и выберите строку Google APIs, как показано на рис. 33.2. Щелкните на кнопке Install 1 package.

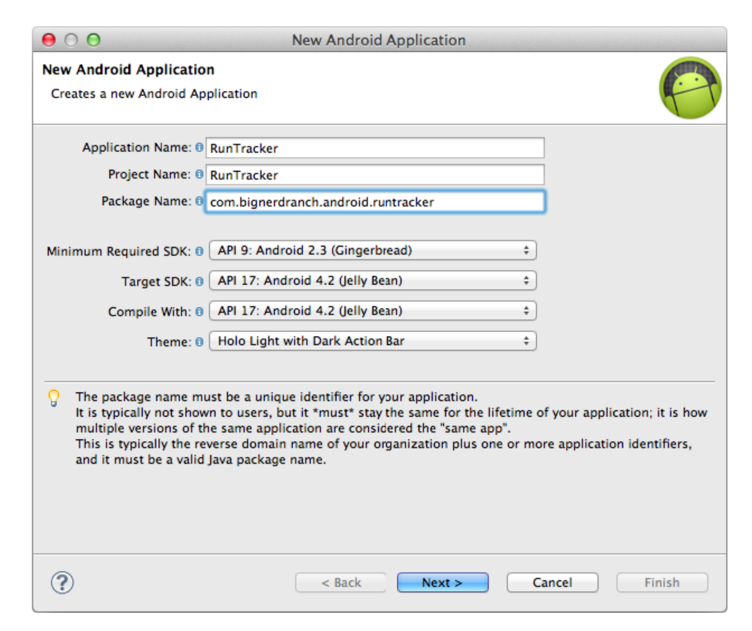


Рис. 33.1. Создание приложения RunTracker

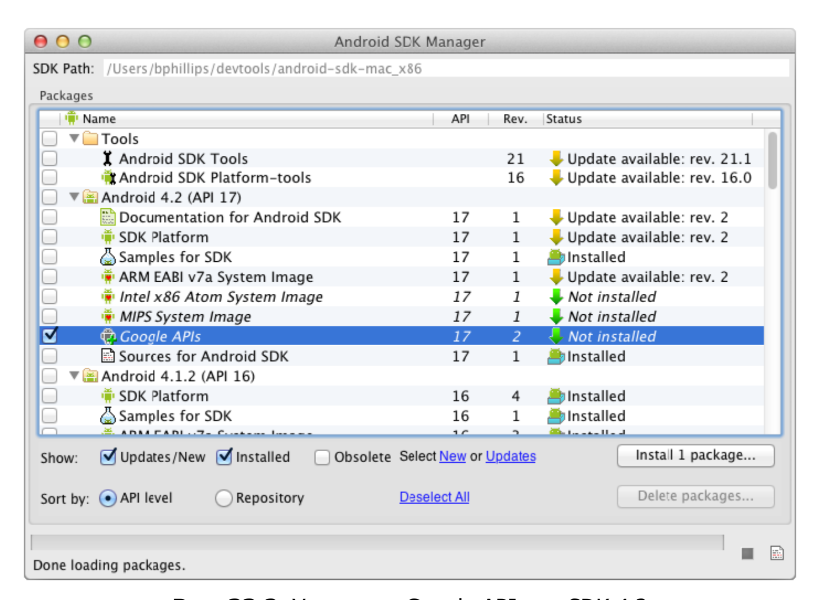


Рис. 33.2. Установка Google API для SDK 4.2

После завершения установки в поле выбора версии SDK появится возможность выбора Google APIs.

Как и в других проектах, прикажите мастеру создать пустую активность. Присвойте ей имя `RunActivity`.

Создание класса RunActivity

RunActivity (и другие активности в RunTracker) будет использовать класс SingleFragmentActivity. Скопируйте файл SingleFragmentActivity.java в пакет com.bignerdranch.android.runtracker, а файл activity_fragment.xml — в res/layout/.

Затем откройте файл RunActivity.java и преобразуйте RunActivity в субкласс SingleFragmentActivity, являющийся хостом для RunFragment. Класс RunFragment еще не существует, но вскоре мы создадим его.

Листинг 33.1. Исходная версия RunActivity (RunActivity.java)

```
public class RunActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new RunFragment();
    }

}
```

Класс RunFragment

На следующем шаге мы создадим пользовательский интерфейс и исходную версию RunFragment. Пользовательский интерфейс RunFragment (рис. 33.3) будет отвечать за вывод простых данных о текущей «серии» и ее местонахождении. В нем имеются кнопки для запуска и остановки текущей серии.



Рис. 33.3. Исходный пользовательский интерфейс RunTracker

Добавление строк

Начнем с добавления ресурсов для строк, представленных на рис. 33.3. Добавьте в файл res/values/strings.xml следующие строки, включая три, которые понадобятся нам позднее в этой главе.

Листинг 33.2. Строки RunTracker (strings.xml)

```
<resources>
  <string name="app_name">RunTracker</string>
  <string name="started">Started:</string>
  <string name="latitude">Latitude:</string>
  <string name="longitude">Longitude:</string>
  <string name="altitude">Altitude:</string>
  <string name="elapsed_time">Elapsed Time:</string>
  <string name="start">Start</string>
  <string name="stop">Stop</string>
  <string name="gps_enabled">GPS Enabled</string>
  <string name="gps_disabled">GPS Disabled</string>
  <string name="cell_text">Run at %1$s</string>
</resources>
```

Файл макета

В макете мы используем виджет `TableLayout`, чтобы аккуратно разместить элементы интерфейса на экране. `TableLayout` состоит из пяти виджетов `TableRow` и одного `LinearLayout`. Каждый виджет `TableRow` содержит два виджета `TextView`: в первом выводится метка, а второй заполняется данными во время выполнения. `LinearLayout` содержит два виджета `Button`.

В этом макете нет ничего, с чем бы мы не работали ранее. Вместо того чтобы создавать интерфейс «с нуля», возьмите готовый макет из архива решений (*http://www.bignerdranch.com/solutions/AndroidProgramming.zip*). Найдите файл 33_Location/RunTracker/res/layout/fragment_run.xml и скопируйте его в каталог res/layout своего проекта.

Создание класса RunFragment

Переходим к созданию самого класса `RunFragment`. Исходная версия будет ограничиваться выводом пользовательского интерфейса и предоставлением доступа к его виджетам.

Листинг 33.3. Заготовка RunFragment (RunFragment.java)

```
public class RunFragment extends Fragment {
    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_run, container, false);

        mStartedTextView = (TextView)view.findViewById(R.id.run_startedTextView);
        mLatitudeTextView = (TextView)view.findViewById(R.id.run_latitudeTextView);
        mLongitudeTextView =
                (TextView)view.findViewById(R.id.run_longitudeTextView);
        mAltitudeTextView = (TextView)view.findViewById(R.id.run_altitudeTextView);
        mDurationTextView = (TextView)view.findViewById(R.id.run_durationTextView);

        mStartButton = (Button)view.findViewById(R.id.run_startButton);

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);

        return view;
    }
}
```

Запустите приложение и убедитесь в том, что оно выглядит так, как показано на рис. 33.3.

Местоположение и LocationManager

Разобравшись с общей структурой, можно переходить к содержательной части. Данные местоположения в Android предоставляются системной службой `LocationManager`. Эта служба предоставляет обновленную позиционную информацию всем приложениям, для которых она представляет интерес. Доставка обновлений осуществляется одним из двух способов.

Первый (и пожалуй, наиболее прямолинейный) способ доставки проходит через интерфейс `LocationListener`. Этот интерфейс предоставляет информацию об обновлениях местоположения (при помощи `onLocationChanged(Location)`), а также обновления состояния и оповещения, включенные или отключенные поставщиком данных.

Использование `LocationListener` для получения обновлений удобно в тех случаях, когда данные местоположения должны поступать только одному компоненту вашего приложения. Например, если бы мы хотели только выводить позиционную информацию в `RunFragment`, можно было бы включить реализацию `LocationListener` в вызов одного из методов `requestLocationUpdates(…)` и `requestSingleUpdate(…)` класса `LocationManager` и считать задачу решенной.

Однако в нашем примере этого недостаточно. Приложение RunTracker должно отслеживать местоположение пользователя независимо от состояния (или наличия) пользовательского интерфейса. Можно было бы воспользоваться закрепляемой службой, но такое решение создает свои проблемы, к тому же в данной ситуации оно выглядит немного тяжеловесно. Лучше использовать интерфейс `PendingIntent` API, появившийся в Android 2.3 (Gingerbread).

Запрашивая информацию местоположения с использованием PendingIntent, вы фактически приказываете LocationManager отправлять вам некую разновидность Intent в будущем. Таким образом, компоненты приложения (и даже весь процесс) могут прекратить существование, а LocationManager будет доставлять интенты, пока вы не прикажете ему остановиться, запустив новые компоненты, которые отреагируют на интенты нужным образом. Например, такая схема позволит предотвратить поглощение приложением лишних ресурсов, в то время как оно активно отслеживает местоположение устройства.

Чтобы управлять взаимодействием с LocationManager, а также дополнительной информацией о текущей серии (см. далее), создайте синглетный класс RunManager, приведенный в листинге 33.4.

Листинг 33.4. Заготовка синглетного класса RunManager (RunManager.java)

```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;

    // Закрытый конструктор заставляет использовать
    // RunManager.get(Context)
    private RunManager(Context appContext) {
        mAppContext = appContext;
        mLocationManager = (LocationManager)mAppContext
            .getSystemService(Context.LOCATION_SERVICE);
    }

    public static RunManager get(Context c) {
        if (sRunManager == null) {
            // Использование контекста приложения для предотвращения
            // утечки активностей
            sRunManager = new RunManager(c.getApplicationContext());
        }
        return sRunManager;
    }

    private PendingIntent getLocationPendingIntent(boolean shouldCreate) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        int flags = shouldCreate ? 0 : PendingIntent.FLAG_NO_CREATE;
        return PendingIntent.getBroadcast(mAppContext, 0, broadcast, flags);
    }

    public void startLocationUpdates() {
        String provider = LocationManager.GPS_PROVIDER;

        // Запуск обновлений из LocationManager
        PendingIntent pi = getLocationPendingIntent(true);
        mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
    }
```

```
    public void stopLocationUpdates() {
        PendingIntent pi = getLocationPendingIntent(false);
        if (pi != null) {
            mLocationManager.removeUpdates(pi);
            pi.cancel();
        }
    }

    public boolean isTrackingRun() {
        return getLocationPendingIntent(false) != null;
    }
}
```

Обратите внимание: класс `RunManager` содержит три открытых метода экземпляров, образующих его базовый API. Он может запускать обновления данных местоположения, останавливать их и сообщать, отслеживается ли в настоящее время серия (а проще говоря, что текущие обновления регистрируются с использованием `LocationManager`).

В `startLocationUpdates()` мы приказываем `LocationManager` передавать обновленную информацию местоположения через поставщика данных GPS как можно чаще. Методу `requestLocationUpdates(String, long, float, PendingIntent)` в параметрах передается минимальная продолжительность ожидания (в миллисекундах) и минимальное смещение (в метрах) перед отправкой следующего обновления.

Эти параметры должны быть настроены на максимальное значение, которое выдержит ваше приложение без ухудшения качества обслуживания. В RunTracker пользователь хочет точно знать, где он находится и где он находился прежде, с максимально возможной точностью. Вот почему мы жестко запрограммировали поставщика данных GPS и запросили максимальную частоту обновлений.

С другой стороны, если приложению достаточно хотя бы в общих чертах представлять, где находится пользователь, большие значения сработают нормально и сэкономят заряд батареи.

Закрытый метод `getLocationPendingIntent(boolean)` создает объект `Intent`, который будет рассылаться при обновлении местоположения. Имя действия идентифицирует событие в нашем приложении, а аргумент `shouldCreate` сообщает `PendingIntent.getBroadcast(…)` (посредством флагов), нужно ли создавать новый экземпляр `PendingIntent` в системе или нет.

Таким образом, реализация метода `isTrackingRun()` — вызов `getLocationPendingIntent(false)` и проверка результата на `null` — определяет, зарегистрирован ли объект `PendingIntent` в ОС.

Получение широковещательных обновлений местоположения

Итак, у нас есть код запроса обновлений местоположения в форме широковещательных интентов. Теперь необходимо реализовать механизм их получения. Приложение RunTracker должно быть способно получать обновления независимо от наличия

или отсутствия, поэтому лучшим местом для их обработки будет автономная реализация BroadcastReceiver, зарегистрированная в манифесте.

Чтобы не усложнять код, мы создадим класс LocationReceiver для регистрации полученных данных местоположения.

Листинг 33.5. Базовая реализация LocationReceiver (LocationReceiver.java)

```
public class LocationReceiver extends BroadcastReceiver {
  private static final String TAG = "LocationReceiver";

  @Override
  public void onReceive(Context context, Intent intent) {
      // Если имеется дополнение Location, использовать его
      Location loc = (Location)intent
          .getParcelableExtra(LocationManager.KEY_LOCATION_CHANGED);
      if (loc != null) {
          onLocationReceived(context, loc);
          return;
      }
      // Если мы попали в эту точку, произошло что-то другое
      if (intent.hasExtra(LocationManager.KEY_PROVIDER_ENABLED)) {
          boolean enabled = intent
              .getBooleanExtra(LocationManager.KEY_PROVIDER_ENABLED, false);
          onProviderEnabledChanged(enabled);
      }
  }

  protected void onLocationReceived(Context context, Location loc) {
      Log.d(TAG, this + " Got location from " + loc.getProvider() + ": "
          + loc.getLatitude() + ", " + loc.getLongitude());
  }

  protected void onProviderEnabledChanged(boolean enabled) {
      Log.d(TAG, "Provider " + (enabled ? "enabled" : "disabled"));
  }

}
```

Как видно из реализации onReceive(Context, Intent), LocationManager упаковывает в интент некоторые дополнения с полезной информацией. Ключ LocationManager.KEY_LOCATION_CHANGED может задавать экземпляр Location, представляющий последнее обновление. Если он возвращается, мы вызываем метод onLocationReceived(Context, Location) для вывода в журнале имени поставщика данных, широты и долготы.

LocationManager также может передавать дополнение логического типа с ключом KEY_PROVIDER_ENABLED; в этом случае вызов onProviderEnabled(boolean) регистрирует этот факт. Вскоре мы субклассируем LocationReceiver, чтобы эти два метода решали более полезные задачи.

Добавьте запись LocationReceiver в манифест приложения RunTracker. Заодно добавьте разрешение ACCESS_FINE_LOCATION и элемент uses-feature для оборудования GPS.

Листинг 33.6. Добавление разрешения на получение данных местонахождения (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.runtracker"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-feature android:required="true"
    android:name="android.hardware.location.gps"/>

  <application android:label="@string/app_name"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:theme="@style/AppTheme">
    <activity android:name=".RunActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <receiver android:name=".LocationReceiver"
      android:exported="false">
      <intent-filter>
        <action android:name="com.bignerdranch.android.runtracker.ACTION_
LOCATION"/>
      </intent-filter>
    </receiver>

  </application>

</manifest>
```

У нас имеется весь служебный код, необходимый для запроса и получения обновленных данных местоположения. Остается организовать пользовательский интерфейс для запуска, остановки и вывода этих данных.

Обновление пользовательского интерфейса данными местоположения

Чтобы убедиться, что все работает как положено, добавьте для кнопок Start и Stop слушателей щелчков, взаимодействующих с `RunManager`. Также добавьте вызовы простого метода `updateUI()`.

Листинг 33.7. Запуск и остановка обновлений местоположения (RunFragment.java)

```
public class RunFragment extends Fragment {
    private RunManager mRunManager;

    private Button mStartButton, mStopButton;
```

продолжение ⇨

Листинг 33.7 (*продолжение*)

```
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.startLocationUpdates();
                updateUI();
            }
        });

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.stopLocationUpdates();
                updateUI();
            }
        });

        updateUI();

        return view;
    }

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
    }
}
```

Если запустить RunTracker с этими добавлениями, вы увидите, как в данных LogCat появляются обновленные данные местоположения.

Для получения улучшенных результатов используйте окно Emulator Control в DDMS для отправки фиктивных обновлений эмулятору или выйдите с устройством на улицу и дождитесь подключения к GPS. Возможно, вам придется несколько минут ожидать первого обновления. Если вам не терпится увидеть результаты или вы заперты в секретном подземном бункере для программистов, через несколько

страниц будет приведена более подробная информация о передаче тестовых данных местоположения.

Вывод данных в LogCat — не самый удобный способ получения информации местоположения. Чтобы вывести данные на экран, реализуйте в `RunFragment` субкласс `LocationReceiver`, который будет сохранять `Location` и обновлять пользовательский интерфейс. Дополнительные данные, хранимые в экземпляре `Run`, позволят вывести начальную дату и продолжительность текущей серии. Начните с реализации простого класса `Run`, который хранит начальную дату, а также умеет вычислять свою продолжительность и форматировать ее в строковом виде.

Листинг 33.8. Класс Run (Run.java)

```
public class Run {
    private Date mStartDate;

    public Run() {
        mStartDate = new Date();
    }

    public Date getStartDate() {
        return mStartDate;
    }

    public void setStartDate(Date startDate) {
        mStartDate = startDate;
    }

    public int getDurationSeconds(long endMillis) {
        return (int)((endMillis - mStartDate.getTime()) / 1000);
    }

    public static String formatDuration(int durationSeconds) {
        int seconds = durationSeconds % 60;
        int minutes = ((durationSeconds - seconds) / 60) % 60;
        int hours = (durationSeconds - (minutes * 60) - seconds) / 3600;
        return String.format("%02d:%02d:%02d", hours, minutes, seconds);
    }

}
```

Используем класс `Run` в сочетании с обновлениями `RunFragment`.

Листинг 33.9. Вывод обновленных данных местоположения (RunFragment.java)

```
public class RunFragment extends Fragment {

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {

        @Override
        protected void onLocationReceived(Context context, Location loc) {
            mLastLocation = loc;
            if (isVisible())
                updateUI();
        }
```

продолжение ⇨

Листинг 33.9 (*продолжение*)

```
        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }

    };

    private RunManager mRunManager;

    private Run mRun;
    private Location mLastLocation;
    private Button mStartButton, mStopButton;

    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.startLocationUpdates();
                mRun = new Run();
                updateUI();
            }
        });

        ...
    }

    @Override
    public void onStart() {
        super.onStart();
        getActivity().registerReceiver(mLocationReceiver,
                new IntentFilter(RunManager.ACTION_LOCATION));
    }

    @Override
    public void onStop() {
        getActivity().unregisterReceiver(mLocationReceiver);
        super.onStop();
    }

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();

        if (mRun != null)
            mStartedTextView.setText(mRun.getStartDate().toString());

        int durationSeconds = 0;
        if (mRun != null && mLastLocation != null) {
         durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
```

```
            mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
            mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
            mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
        }
        mDurationTextView.setText(Run.formatDuration(durationSeconds));

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
    }

}
```

Прежде всего обратите внимание на новые переменные экземпляров для `Run` и последнего полученного объекта `Location`. Эти данные используются при выполнении обнулений пользовательского интерфейса, выполняемых в `updateUI()`. Объект `Run` инициализируется при включении обновления данных местоположения.

Мы создаем анонимный класс `LocationReceiver` и присваиваем его `mLocationReceiver`, чтобы сохранить полученную позицию и обновить пользовательский интерфейс. Также на экране отображается сообщение с информацией о том, включен поставщик данных GPS или нет.

Наконец, реализации методов `onStart()` и `onStop()` используются для регистрации и отмены регистрации получателя в сочетании с фрагментом, видимым пользователю. Эти операции также могут выполняться в методах `onCreate(Bundle)` и `onDestroy()`, чтобы переменная `mLastLocation` всегда содержала последнее обновление местоположения, даже если в момент его получения фрагмент находился вне экрана.

Снова запустите приложение RunTracker. На этот раз пользовательский интерфейс должен заполняться реальными данными местоположения.

Ускорение отклика: последнее известное местоположение

В некоторых ситуациях пользователю совершенно не хочется несколько минут ждать, пока ваше устройство свяжется с таинственными спутниками в космосе, чтобы узнать, где он находится. К счастью, мы можем избавить его от ожидания, используя последнее известное местоположение `LocationManager` для любого поставщика позиционных данных.

В нашем приложении используется только поставщик GPS, а его последнее местоположение запрашивается достаточно элементарно. Остается лишь как-то передать информацию в пользовательский интерфейс, но для этого можно просто использовать широковещательную рассылку `Intent` как от лица `LocationManager`.

Листинг 33.10. Получение последнего известного местоположения (RunManager.java)

```
public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;

    // Получение последнего известного местоположения
    // и его рассылка (если данные доступны).
```

продолжение ⇗

Листинг 33.10 (*продолжение*)

```
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Время инициализируется текущим значением
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }

    // Запуск получения обновлений от LocationManager
    PendingIntent pi = getLocationPendingIntent(true);
    mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
}

private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mAppContext.sendBroadcast(broadcast);
}
```

Обратите внимание на сброс временной метки местоположения, полученного от поставщика данных GPS. Возможно, это именно то, что нужно пользователю — а может, и нет; мы оставляем это решение читателю для самостоятельной работы.

У объекта `LocationManager` можно запросить данные последнего известного местоположения для любого известного ему поставщика. Также можно запросить список всех известных поставщиков методом `getAllProviders()`. Например, вы можете перебрать все известные последние местоположения, проверить их точность и по временной метке определить самые свежие данные. Тем самым предотвращается использование устаревших данных.

Тестирование на реальных и виртуальных устройствах

Тестирование таких приложений, как RunTracker, может быть непростым делом даже для опытного программиста. Нужно позаботиться о том, чтобы позиционные данные, полученные от системы, правильно отслеживались и сохранялись. Перемещения усложняют эту задачу даже на небольших скоростях.

В подобных ситуациях можно передавать `LocationManager` тестовые данные, имитирующие нахождение устройства в заданном месте.

Для этого проще всего воспользоваться окном Emulator Control в DDMS. Данное решение работает только для виртуальных устройств, но оно позволяет задавать новые данные местоположения либо вручную (по одному), либо в формате файла GPX или KML с серией точек, которые посещались за некоторый промежуток времени.

Для тестирования системы определения местоположения на реальном устройстве придется дополнительно потрудиться, но и это вполне возможно. Основная схема выглядит так.

1. Запросить разрешение `ACCESS_MOCK_LOCATION`.

2. Добавить тестового поставщика данных методом `LocationManager.addTestProvider(…)`.
3. Разрешить использование поставщика методом `setTestProviderEnabled(…)`.
4. Задать исходный статус поставщика методом `setTestProviderStatus(…)`.
5. Опубликовать данные местоположения методом `setTestProviderLocation(…)`.
6. Удалить тестового поставщика данных методом `removeTestProvider(…)`.

К счастью, мы уже выполнили всю черную работу за вас. На сайте Big Nerd Ranch размещен простой проект TestProvider, который вы можете загрузить, установить на своем устройстве и запустить для управления поставщиком тестовых данных.

Загрузите репозиторий Android Course Resources из хранилища Github по адресу *https://github.com/bignerdranch/AndroidCourseResources* и импортируйте каталог TestProvider как проект в Eclipse.

Чтобы в приложении RunTracker вместо GPS использовался новый поставщик тестовых данных, необходимо внести изменения, представленные в следующем листинге.

Листинг 33.11. Использование тестового поставщика (RunManager.java)

```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;

    ...

    public void startLocationUpdates() {
        String provider = LocationManager.GPS_PROVIDER;
        // Если имеется поставщик тестовых данных и он активен,
        // использовать его.
        if (mLocationManager.getProvider(TEST_PROVIDER) != null &&
                mLocationManager.isProviderEnabled(TEST_PROVIDER)) {
            provider = TEST_PROVIDER;
        }
        Log.d(TAG, "Using provider " + provider);

        // Получить последнее известное местоположение
        // и отправить его, если данные действительны.
        Location lastKnown = mLocationManager.getLastKnownLocation(provider);
        if (lastKnown != null) {
            // Время инициализируется текущим значением
            lastKnown.setTime(System.currentTimeMillis());
            broadcastLocation(lastKnown);
        }
```

Возможно, чтобы проект TestProvider заработал, вам придется установить флажок Allow mock locations в меню Developer options из приложения Settings (рис. 33.4).

Запустите приложение TestProvider на своем устройстве и нажмите кнопку, чтобы запустить фиктивные обновления местоположения.

Затем запустите RunTracker и понаблюдайте за поступлением фиктивных данных. Когда тестирование будет завершено, отключите поставщика тестовых данных, чтобы ваше устройство не путалось в своем текущем местоположении.

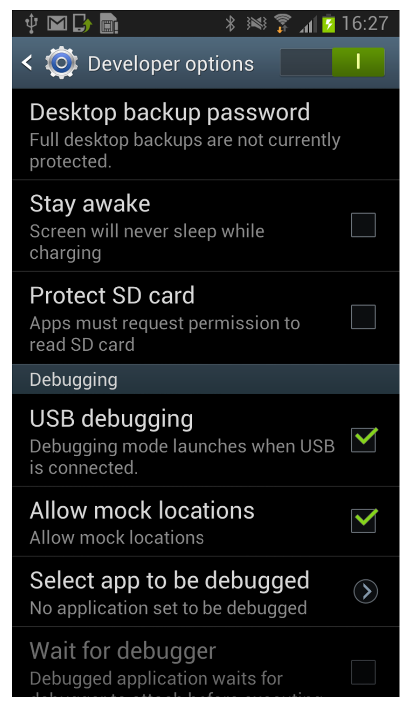


Рис. 33.4. Включение поставки фиктивных данных местоположения

34 Локальные базы данных и SQLite

Приложениям с большими и сложными наборами данных часто оказывается недостаточно возможностей простых файловых форматов JSON. Например, в приложении RunTracker пользователь может бесконечно долго отслеживать свое местоположение, в результате чего будет сгенерирован большой объем данных. В Android для хранения таких наборов обычно используются базы данных SQLite. SQLite — мультиплатформенная библиотека, распространяемая с открытым кодом, которая предоставляет мощный API реляционной базы данных для работы с одним файлом на диске.

В Android Java-интерфейс к SQLite реализован в классе `SQLiteDatabase`, возвращающем наборы данных в виде экземпляров `Cursor`. В этой главе мы создадим для RunTracker механизм хранения данных, использующий базу данных для хранения информации о сериях и их позиций. Также будет создана новая активность со списком серий и фрагмент для создания и отслеживания серий.

Хранение серий и позиций в базе данных

Чтобы сохранить какие-либо данные в базе, сначала необходимо определить структуру базы данных и открыть ее. Поскольку эта задача весьма типична для Android, для нее существует вспомогательный класс. `SQLiteOpenHelper` инкапсулирует служебные операции по созданию, открытию и обновлению баз данных для хранения данных приложения.

В RunTracker мы создадим субкласс `SQLiteOpenHelper` с именем `RunDatabaseHelper`. Объект `RunManager` будет содержать закрытый экземпляр `RunDatabaseHelper` и предоставлять приложению API для вставки, обработки запросов и других операций с данными в базе. Класс `RunDatabaseHelper` предоставляет методы, которые `RunManager` будет вызывать для реализации большей части своего API.

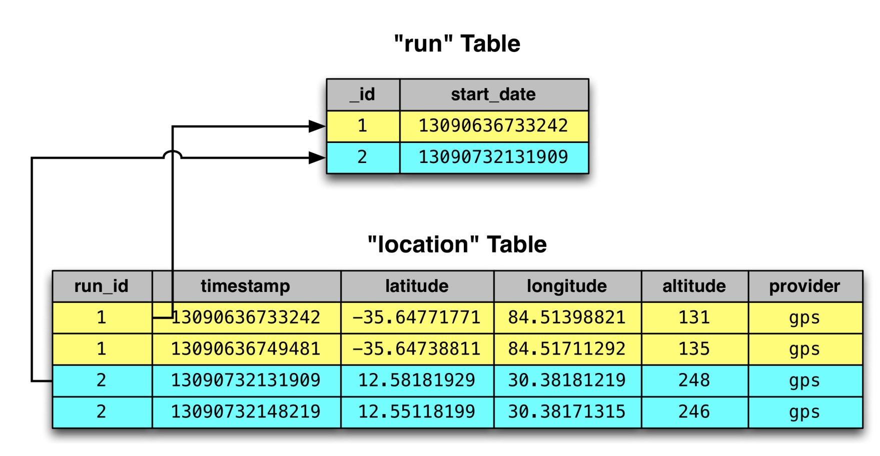
При проектировании API хранения информации в базе данных разработчик обычно создает по одному субклассу `SQLiteOpenHelper` для каждого вида баз данных,

которыми он собирается управлять. Затем для каждого отдельного файла базы данных SQKite создается один экземпляр этого субкласса. В большинстве приложений, включая RunTracker, создается всего один субкласс `SQLiteOpenHelper`, а его единственный экземпляр используется совместно всеми компонентами приложения.

В объектно-ориентированном программировании при проектировании баз данных чаще всего для каждого класса в модели данных приложения создается отдельная таблица. В приложении RunTracker в базе данных должны храниться данные двух классов: `Run` и `Location`.

Соответственно в нашем примере будут созданы две таблицы: `run` (для серий) и `location` (для данных местоположения). Объект `Run` может содержать много объектов `Location`, поэтому в таблицу `location` будет включен столбец внешнего ключа `run_id`, ссылающийся на столбец `_id` таблицы `run`.

Структура таблиц изображена на рис. 34.1.



"run" Table

_id	start_date
1	13090636733242
2	13090732131909

"location" Table

run_id	timestamp	latitude	longitude	altitude	provider
1	13090636733242	-35.64771771	84.51398821	131	gps
1	13090636749481	-35.64738811	84.51711292	135	gps
2	13090732131909	12.58181929	30.38181219	248	gps
2	13090732148219	12.55118199	30.38171315	246	gps

Рис. 34.1. Схема базы данных RunTracker

Создайте класс `RunDatabaseHelper` и введите код, приведенный в листинге 34.1.

Листинг 34.1. Исходная версия RunDatabaseHelper (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    public RunDatabaseHelper(Context context) {
        super(context, DB_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Создание таблицы "run"
        db.execSQL("create table run (" +
```

```
            "_id integer primary key autoincrement, start_date integer)");
        // Создание таблицы "location"
        db.execSQL("create table location (" +
            " timestamp integer, latitude real, longitude real, altitude real," +
            " provider varchar(100), run_id integer references run(_id))");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Здесь реализуются изменения схемы и преобразования данных
        // при обновлении схемы
    }

    public long insertRun(Run run) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());
        return getWritableDatabase().insert(TABLE_RUN, null, cv);
    }
}
```

Реализация субкласса SQLiteOpenHelper требует переопределения двух методов: onCreate(SQLiteDatabase) и onUpgrade(SQLiteDatabase, int, int). В методе onCreate(…) мы устанавливаем схему создаваемой базы данных, а в методе onUpgrade(…) можно разместить код перехода между версиями схемы базы данных.

Также на практике часто реализуется упрощенный конструктор, который заполняет значения некоторых аргументов, необходимых версии из суперкласса. В нашем примере передается постоянное имя файла базы данных, null вместо необязательного объекта CursorFactory и постоянный целочисленный номер версии.

Хотя в примере RunTracker это и не требуется, SQLiteOpenHelper поддерживает возможность управления разными версиями схемы базы данных. Предполагается, что номера версий представляют собой целые числа, начиная с 1. В реальном приложении при каждом изменении схемы базы данных разработчик увеличивает константу версии и включает в метод onUpgrade(…) код управления изменениями схемы или данных, необходимыми между версиями.

В нашей реализации метода onCreate(…) с только что созданной базой данных выполняются две инструкции SQL CREATE TABLE. Также реализован метод insertRun(Run), который вставляет новую строку в таблицу run и возвращает ее идентификатор. Запись серии содержит всего одно поле данных — начальную дату; ее значение в формате long сохраняется в базе данных с использованием объекта ContentValues, представляющего соответствие между именами столбцов и значениями.

Класс SQLiteOpenHelper содержит два метода, предоставляющих доступ к экземпляру SQLiteDatabase: getWritableDatabase() и getReadableDatabase(). Если требуется база данных с возможностью записи, используется метод getWritableDatabase(), а если достаточно доступа только для чтения — getReadableDatabase(). На практике реализации этих методов возвращают один экземпляр SQLiteDatabase для заданного экземпляра SQLiteOpenHelper, но в некоторых редких ситуациях (например, при заполнении диска) можно получить базу данных, доступную для чтения, в ситуации, в которой невозможно получить базу данных с возможностью записи.

Чтобы обеспечить возможность запроса одной или нескольких серий из базы данных и различать их в приложении, в класс Run необходимо добавить свойство-идентификатор. Внесите изменения, представленные в листинге 34.2.

Листинг 34.2. Включение идентификатора в класс Run

```
public class Run {
    private long mId;
    private Date mStartDate;

    public Run() {
        mId = -1;
        mStartDate = new Date();
    }

    public long getId() {
        return mId;
    }

    public void setId(long id) {
        mId = id;
    }

    public Date getStartDate() {
        return mStartDate;
    }
```

Затем необходимо доработать класс RunManager, чтобы он использовал новую базу данных. Так будет создан API, который будет использоваться остальным кодом приложения для хранения и выборки данных. Для начала ограничимся минимальным кодом, обеспечивающим сохранение объектов Run.

Листинг 34.3. Управление текущей серией (RunManager.java)

```
public class RunManager {
    private static final String TAG = "RunManager";

    private static final String PREFS_FILE = "runs";
    private static final String PREF_CURRENT_RUN_ID = "RunManager.currentRunId";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;
    private RunDatabaseHelper mHelper;
    private SharedPreferences mPrefs;
    private long mCurrentRunId;

    private RunManager(Context appContext) {
        mAppContext = appContext;
        mLocationManager = (LocationManager)mAppContext
            .getSystemService(Context.LOCATION_SERVICE);
        mHelper = new RunDatabaseHelper(mAppContext);
        mPrefs = mAppContext.getSharedPreferences(PREFS_FILE, Context.MODE_PRIVATE);
```

```
        mCurrentRunId = mPrefs.getLong(PREF_CURRENT_RUN_ID, -1);
    }

    ...

    private void broadcastLocation(Location location) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
        mAppContext.sendBroadcast(broadcast);
    }

    public Run startNewRun() {
        // Вставка объекта Run в базу данных
        Run run = insertRun();
        // Запуск отслеживания серии
        startTrackingRun(run);
        return run;
    }

    public void startTrackingRun(Run run) {
        // Получение идентификатора
        mCurrentRunId = run.getId();
        // Сохранение его в общих настройках
        mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();
        // Запуск обновления данных местоположения
        startLocationUpdates();
    }
   public void stopRun() {
        stopLocationUpdates();
        mCurrentRunId = -1;
        mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();
    }
    private Run insertRun() {
        Run run = new Run();
        run.setId(mHelper.insertRun(run));
        return run;
    }
}
```

Присмотримся к новым методам, добавленным в RunManager. Метод startNewRun() вызывает insertRun() для создания и вставки в базу данных нового объекта Run, передает его при вызове startTrackingRun(Run) для начала отслеживания и, наконец, возвращает его вызывающей стороне. Этот метод будет использоваться классом RunFragment в ответ на нажатие кнопки Start при отсутствии текущей серии.

RunFragment также будет напрямую вызывать startTrackingRun(Run) при перезапуске отслеживания для текущей серии. Этот метод сохраняет идентификатор переданного ему объекта Run в переменной экземпляра и в общих настройках. При таком способе сохранения данные можно будет получить позднее даже при полном уничтожении приложения; в этом случае необходимую работу выполнит конструктор RunManager.

Наконец, метод stopRun() останавливает обновления и стирает идентификатор текущей серии. RunFragment использует этот метод в реализации кнопки Stop.

Раз уж мы взялись за RunFragment, сейчас будет уместно использовать новые методы RunManager. Внесите изменения, представленные в листинге 34.4.

Листинг 34.4. Обновление кода запуска и остановки (RunFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);
    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}
```

Затем нам понадобится реализовать вставку объектов `Location` в базу данных в ответ на обновления от `LocationManager`. По аналогии со вставкой объектов `Run`, мы добавим в `RunDatabaseHelper` и `RunManager` методы для вставки позиции в текущую серию. Однако в отличие от `Run` класс `RunTracker` должен быть способен вставлять данные позиций по мере поступления обновлений независимо от того, отображается ли пользовательский интерфейс и работает ли приложение. Для реализации этого требования лучше всего использовать автономный объект `BroadcastReceiver`.

Начнем с добавления метода `insertLocation(long, Location)` в `RunDatabaseHelper`.

Листинг 34.5. Вставка позиций в базу данных (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    private static final String TABLE_LOCATION = "location";
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";
```

```
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";
    private static final String COLUMN_LOCATION_PROVIDER = "provider";
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";
    ...

    public long insertLocation(long runId, Location location) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());
        cv.put(COLUMN_LOCATION_RUN_ID, runId);
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);
    }
```

Теперь добавьте в `RunManager` код вставки позиции для текущей серии.

Листинг 34.6. Вставка позиции для текущей серии (RunManager.java)

```
    private Run insertRun() {
        Run run = new Run();
        run.setId(mHelper.insertRun(run));
        return run;
    }

    public void insertLocation(Location loc) {
        if (mCurrentRunId != -1) {
            mHelper.insertLocation(mCurrentRunId, loc);
        } else {
            Log.e(TAG, "Location received with no tracking run; ignoring.");
        }
    }
}
```

И наконец, мы должны найти подходящее место для вызова нового метода `insertLocation(Location)`. Использование для этой цели автономного объекта `BroadcastReceiver` гарантирует, что интенты позиций будут обработаны независимо от работоспособности остальных компонентов приложения RunTracker. Для этого необходимо создать специализированный субкласс `LocationReceiver` с именем `TrackingLocationReceiver` и зарегистрировать его с использованием фильтра интентов в манифесте.

Создайте `TrackingLocationReceiver` как класс верхнего уровня.

Листинг 34.7. Класс TrackingLocationReceiver: просто, но элегантно (TrackingLocationReceiver.java)

```
public class TrackingLocationReceiver extends LocationReceiver {

    @Override
    protected void onLocationReceived(Context c, Location loc) {
        RunManager.get(c).insertLocation(loc);
    }
}
```

Зарегистрируйте его в манифесте, чтобы его код выполнялся в ответ на специализированное действие ACTION_LOCATION.

Листинг 34.8. Включение TrackingLocationReceiver (AndroidManifest.xml)

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">

  ...

  <receiver android:name=".LocationReceiver"
  <receiver android:name=".TrackingLocationReceiver"
    android:exported="false">
    <intent-filter>
      <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
    </intent-filter>
  </receiver>

</application>
```

После всех изменений запустите приложение и убедитесь в том, что ваша тяжелая работа не пропала даром. Приложение RunTracker теперь умеет отслеживать серии до тех пор, пока вы не прикажете ему остановиться, даже в случае уничтожения или завершения приложения. Чтобы убедиться в том, что все работает, как задумано, добавьте команду регистрации в «успешную» ветку метода onLocationReceived(…) класса TrackingLocationReceiver. Запустите серию, затем уничтожьте или завершите приложение, продолжая наблюдать за данными LogCat.

Запрос списка серий из базы данных

Приложение RunTracker теперь может сохранять в базе данных новые серии и их позиции, но в своей текущей версии RunFragment создает новую серию при каждом нажатии кнопки Start. В этом разделе мы добавим новую активность и фрагмент для вывода списка серий. С их помощью пользователь сможет создавать новые и просматривать существующие серии. Этот интерфейс напоминает тот, который мы реализовали в CriminalIntent для списка преступлений, но только данные, на основе которых строится список, берутся из базы данных, а не из памяти RunTracker и хранилища файловой системы.

Запрос к SQLiteDatabase возвращает объект курсора Cursor, описывающий результат. API курсоров достаточно прост и гибок, чтобы поддерживать любые виды разновидности результатов от любого запроса. Курсоры рассматривают свои результаты как совокупность строк и столбцов, а в значениях могут использоваться только примитивные типы и String.

Впрочем, Java-программисты привыкли использовать для инкапсуляции модели данных объекты — например, Run и Location. Поскольку у нас уже имеются готовые таблицы базы данных, представляющие объекты, было бы идеально, если бы мы могли получать от Cursor экземпляры этих объектов.

У Cursor имеется встроенный субкласс с именем CursorWrapper, который инкапсулирует существующий экземпляр Cursor и передает ему все вызовы методов. Сам по себе он не очень полезен, но при использовании в качестве суперкласса предоставляет хорошую основу для построения собственных реализаций курсоров для объектов модели.

Обновите класс RunDatabaseHelper и включите в него новый метод queryRuns(), который возвращает объект RunCursor со списком всех серий, упорядоченным по дате.

Листинг 34.9. Запрос списка серий (RunDatabaseHelper.java)

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_ID = "_id";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    ...

    public RunCursor queryRuns() {
        // Эквивалент "select * from run order by start_date asc"
        Cursor wrapped = getReadableDatabase().query(TABLE_RUN,
                null, null, null, null, null, COLUMN_RUN_START_DATE + " asc");
        return new RunCursor(wrapped);
    }

    /**
     * Вспомогательный класс с курсором, возвращающим строки таблицы "run".
     * Метод {@link getRun()} возвращает экземпляр Run, представляющий
     * текущую строку.
     */
    public static class RunCursor extends CursorWrapper {

        public RunCursor(Cursor c) {
            super(c);
        }

        /**
         * Возвращает объект Run, представляющий текущую строку,
         * или null, если текущая строка недействительна.
         */
        public Run getRun() {
            if (isBeforeFirst() || isAfterLast())
                return null;
            Run run = new Run();
            long runId = getLong(getColumnIndex(COLUMN_RUN_ID));
            run.setId(runId);
            long startDate = getLong(getColumnIndex(COLUMN_RUN_START_DATE));
            run.setStartDate(new Date(startDate));
            return run;
        }
    }
}
```

Класс RunCursor определяет всего два метода: простой конструктор и getRun(). Метод getRun() проверяет, что курсор находится в своих границах, а затем создает

и настраивает экземпляр Run по значениям в столбцах текущей строки. Пользователь RunCursor перебирает строки полученного набора и вызывает getRun() для каждой строки, чтобы получить удобный объект вместо кучи неудобных примитивов.

Соответственно главной целью RunCursor является инкапсуляция рутинной работы по преобразованию строки таблицы run в экземпляр Run, со всеми необходимыми перемещениями и преобразованиями данных.

Метод queryRuns() выполняет всю работу по выполнению запроса SQL и получению нового экземпляра RunCursor, который возвращается вызывающей стороне. Теперь этот новый метод можно будет использовать в RunManager, а затем и в RunListFragment.

Листинг 34.10. Опосредованное выполнение запросов данных серий (RunManager.java)

```
    private Run insertRun() {
        Run run = new Run();
        run.setId(mHelper.insertRun(run));
        return run;
    }

    public RunCursor queryRuns() {
        return mHelper.queryRuns();
    }

    public void insertLocation(Location loc) {
        if (mCurrentRunId != -1) {
            mHelper.insertLocation(mCurrentRunId, loc);
        } else {
            Log.e(TAG, "Location received with no tracking run; ignoring.");
        }
    }
```

Вывод списка серий с использованием CursorAdapter

Чтобы заложить основу для пользовательского интерфейса списка серий, создайте новую активность RunListActivity и назначьте ее активностью по умолчанию в манифесте. На ошибку с RunListFragment пока не обращайте внимания.

Листинг 34.11. Класс RunListActivity (RunListActivity.java)

```
public class RunListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new RunListFragment();
    }

}
```

Листинг 34.12. Настройка RunListActivity (AndroidManifest.xml)

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">
  <activity android:name=".RunActivity"
  <activity android:name=".RunListActivity"
    android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity android:name=".RunActivity"
            android:label="@string/app_name" />
  <receiver android:name=".TrackingLocationReceiver"
    android:exported="false">
    <intent-filter>
```

Теперь можно переходить к созданию заготовки RunListFragment. Пока мы загружаем курсор в onCreate(Bundle) и закрываем его в onDestroy(), но использовать такую схему не рекомендуется, потому что с ней запрос к базе данных выполняется в главном потоке (UI-потоке), а в некоторых случаях это может привести к выдаче ужасного сообщения ANR (Application Not Responding). В следующей главе эта реализация будет заменена другой, использующей класс Loader для переведения работы в фоновый режим.

Листинг 34.13. Исходная версия RunListFragment (RunListFragment.java)

```
public class RunListFragment extends ListFragment {

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Запрос на получение списка серий
        mCursor = RunManager.get(getActivity()).queryRuns();
    }

    @Override
    public void onDestroy() {
        mCursor.close();
        super.onDestroy();
    }

}
```

От класса RunCursor не будет никакой пользы без механизма передачи данных виджету ListView, связанному с RunListFragment. В Android API (и в библиотеку поддержки) входит класс CursorAdapter, который делает именно то, что нужно. Вам остается лишь субклассировать его и предоставить реализации пары методов. Класс CursorAdapter берет на себя логику создания и повторного использования представлений, так что вам этим заниматься не придется.

Включите в класс `RunListFragment` реализацию `RunCursorAdapter`.

Листинг 34.14. Реализация RunCursorAdapter (RunListFragment.java)

```
public class RunListFragment extends ListFragment {

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Запрос на получение списка серий
        mCursor = RunManager.get(getActivity()).queryRuns();
        // Создание адаптера, ссылающегося на этот курсор
        RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);
        setListAdapter(adapter);
    }

    @Override
    public void onDestroy() {
        mCursor.close();
        super.onDestroy();
    }

    private static class RunCursorAdapter extends CursorAdapter {

        private RunCursor mRunCursor;

        public RunCursorAdapter(Context context, RunCursor cursor) {
            super(context, cursor, 0);
            mRunCursor = cursor;
        }

        @Override
        public View newView(Context context, Cursor cursor, ViewGroup parent) {
            // Использование заполнителя макета для получения
            // представления строки
            LayoutInflater inflater = (LayoutInflater)context
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            return inflater
                .inflate(android.R.layout.simple_list_item_1, parent, false);
        }

        @Override
        public void bindView(View view, Context context, Cursor cursor) {
            // Получение серии для текущей строки
            Run run = mRunCursor.getRun();

            // Создание текстового представления начальной даты
            TextView startDateTextView = (TextView)view;
            String cellText =
                context.getString(R.string.cell_text, run.getStartDate());
            startDateTextView.setText(cellText);
        }
    }
}
```

Конструктор CursorAdapter получает Context, Cursor и целочисленный набор флагов. Многие флаги сейчас считаются устаревшими, или вместо них рекомендуется использовать загрузчики, поэтому в этом аргументе передается нуль. Мы также сохраняем объект RunCursor в переменной экземпляра, чтобы избежать его последующего преобразования.

Затем мы реализуем метод newView(Context, Cursor, ViewGroup), возвращающий объект View для текущей строки курсора. В нашем примере используется заполнение системного ресурса android.R.layout.simple_list_item_1, который представляет собой простой виджет TextView. Поскольку все представления в списке будут выглядеть одинаково, это вся логика, которая нам сейчас понадобится.

Метод bindView(View, Context, Cursor) будет вызываться кодом CursorAdapter, когда последнему потребуется настроить представление для хранения данных строки в курсоре. При вызове всегда передается объект View, возвращенный ранее из newView(…).

Реализация bindView(…) относительно проста. Сначала мы запрашиваем у RunCursor объект Run для текущей строки (курсор уже будет позиционирован CursorAdapter). Затем мы предполагаем, что переданное представление представляет собой TextView, и настраиваем его для вывода простого описания Run.

С этими изменениями запустите приложение RunTracker; на экране выводится список ранее созданных серий (предполагается, что вы запустили приложение и начали новую серию после реализации кода вставки в базу данных, приведенного ранее в этой главе). Если серии еще не создавались, не огорчайтесь: в следующем разделе мы добавим пользовательский интерфейс для создания серий.

Создание новых серий

Пользовательский интерфейс для создания новых серий легко реализуется на панели действий, как это делалось в приложении CriminalIntent. Начните с создания ресурса меню.

Листинг 34.15. Меню списка серий (run_list_options.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/menu_item_new_run"
    android:showAsAction="always"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/new_run"/>
</menu>
```

В меню содержится ссылка на строку; добавим ее в файл strings.xml.

Листинг 34.16. Добавление строки New Run (strings.xml)

```
  <string name="stop">Stop</string>
  <string name="gps_enabled">GPS Enabled</string>
  <string name="gps_disabled">GPS Disabled</string>
  <string name="cell_text">Run at %1$s</string>
    <string name="new_run">New Run</string>
</resources>
```

Добавьте в RunListFragment код создания командного меню и обработки выбора команды, приведенный в следующем листинге.

Листинг 34.17. Новые серии в командном меню (RunListFragment.java)

```
public class RunListFragment extends ListFragment {
    private static final int REQUEST_NEW_RUN = 0;

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        // Запрос на получение списка серий
        mCursor = RunManager.get(getActivity()).queryRuns();
        // Создание адаптера, ссылающегося на этот курсор
        RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);
        setListAdapter(adapter);
    }

    ...

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.run_list_options, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.menu_item_new_run:
            Intent i = new Intent(getActivity(), RunActivity.class);
            startActivityForResult(i, REQUEST_NEW_RUN);
            return true;
        default:
            return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (REQUEST_NEW_RUN == requestCode) {
            mCursor.requery();
            ((RunCursorAdapter)getListAdapter()).notifyDataSetChanged();
        }
    }

    private static class RunCursorAdapter extends CursorAdapter {

        private RunCursor mRunCursor;
```

Единственный новый и интересный момент в этом решении — использование onActivityResult(…) для принудительной перезагрузки списка после того, как пользователь вернется к нему в процессе навигации. И снова повторный запрос

курсора в том виде, в котором он приведен здесь (в главном потоке), оставляет желать лучшего. В следующей главе мы заменим его кодом с использованием Loader.

Работа с существующими сериями

Следующий логичный шаг — переход от списка серий к подробной информации о конкретной серии. Чтобы эта возможность заработала, RunFragment понадобится передавать идентификатор серии в аргументе. Поскольку хостом RunFragement является активность RunActivity, ей тоже понадобится дополнение для идентификатора серии.

Начнем с добавления аргумента в RunFragment и метода newInstance(long), упрощающего его использование.

Листинг 34.18. Добавление аргумента для идентификатора серии (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    ...
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    public static RunFragment newInstance(long runId) {
        Bundle args = new Bundle();
        args.putLong(ARG_RUN_ID, runId);
        RunFragment rf = new RunFragment();
        rf.setArguments(args);
        return rf;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

Теперь используем новые возможности фрагмента в RunActivity. Если интент содержит дополнение RUN_ID, то для создания RunFragment используется метод newInstance(long). В противном случае просто используется конструктор по умолчанию, как и прежде.

Листинг 34.19. Добавление дополнения для идентификатора серии (RunActivity.java)

```
public class RunActivity extends SingleFragmentActivity {
    /** Ключ для передачи идентификатора серии в формате long */
    public static final String EXTRA_RUN_ID =
        "com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment() {
        return new RunFragment();
        long runId = getIntent().getLongExtra(EXTRA_RUN_ID, -1);
        if (runId != -1) {
```

продолжение ⇗

Листинг 34.19 (*продолжение*)

```
            return RunFragment.newInstance(runId);
        } else {
            return new RunFragment();
        }
    }
}
```

Последнее, что осталось сделать, — запрограммировать реакцию на выделение элемента списка в `RunListFragment` запуском `RunActivity` с идентификатором выбранной серии.

Листинг 34.20. Открытие существующих серий через onListItemClick(...) (RunListFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Аргумент id содержит идентификатор серии;
    // CursorAdapter автоматически предоставляет эту информацию.
    Intent i = new Intent(getActivity(), RunActivity.class);
    i.putExtra(RunActivity.EXTRA_RUN_ID, id);
    startActivity(i);
}

private static class RunCursorAdapter extends CursorAdapter {
```

Здесь используется один маленький фокус. Так как мы присвоили столбцу идентификатора в таблице `run` имя `_id`, класс `CursorAdapter` обнаружил этот факт и передал его в аргументе `id` метода `onListItemClick(...)`. Соответственно мы можем передать его непосредственно `RunActivity` в дополнении. Очень удобно!

К сожалению, на этом все удобства кончаются. Простого запуска `RunFragment` с аргументом-идентификатором недостаточно для вывода полезной информации о существующей серии. Необходимо запросить в базе данных подробную информацию, включая последнее сохраненное местоположение, для заполнения пользовательского интерфейса.

К счастью, эта задача достаточно близка к тому, что мы уже сделали. Начать стоит с класса `RunDatabaseHelper`, в котором будет создан новый метод `queryRun(long)`, возвращающий объект `RunCursor` для одной серии по заданному идентификатору.

Листинг 34.21. Запрос одной серии (RunDatabaseHelper.java)

```
public RunCursor queryRun(long id) {
    Cursor wrapped = getReadableDatabase().query(TABLE_RUN,
            null, // Все столбцы
            COLUMN_RUN_ID + " = ?", // Поиск по идентификатору серии
            new String[]{ String.valueOf(id) }, // С этим значением
            null, // group by
            null, // order by
            null, // having
            "1"); // 1 строка
    return new RunCursor(wrapped);
}
```

Смысл многочисленных аргументов метода query(…) становится понятен. Мы выбираем все столбцы из TABLE_RUN и фильтруем их по столбцу идентификатора, который передается в аргументе условия WHERE с использованием массива строк с одним элементом. Запрос ограничивается возвращением одной строки, результат упаковывается в RunCursor и возвращается.

Затем в класс RunManager добавляется метод getRun(long), который упаковывает результат только что созданного метода queryRun(long) и извлекает объект Run из первой строки (если она есть).

Листинг 34.22. Реализация getRun(long) (RunManager.java)

```
public Run getRun(long id) {
    Run run = null;
    RunCursor cursor = mHelper.queryRun(id);
    cursor.moveToFirst();
    // Если строка присутствует, получить объект серии
    if (!cursor.isAfterLast())
        run = cursor.getRun();
    cursor.close();
    return run;
}
```

Этот метод пытается извлечь Run из первой строки набора RunCursor, полученного при вызове queryRun(long). Сначала он приказывает RunCursor перейти к первой строке результата. Если набор содержит хотя бы одну строку, isAfterLast() вернет false, и мы можем спокойно запросить объект Run для этой строки.

Так как для вызывающей стороны нового метода объект RunCursor недоступен, не забудьте вызвать close() перед возвращением, чтобы база данных могла как можно скорее освободить ресурсы курсора в памяти.

Эта часть работы завершена, теперь мы можем обновить RunFragment для работы с существующими сериями.

Внесите изменения, представленные в листинге 34.23.

Листинг 34.23. Работа с существующими сериями (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {

        @Override
        protected void onLocationReceived(Context context, Location loc) {
            if (!mRunManager.isTrackingRun(mRun))
                return;
            mLastLocation = loc;
            if (isVisible())
                updateUI();
        }

        @Override
```

продолжение ⇗

Листинг 34.23 (*продолжение*)

```
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }

    };

    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());<

        // Проверить идентификатор Run и получить объект серии
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            if (runId != -1) {
                mRun = mRunManager.getRun(runId);
            }
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_run, container, false);

        ...

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRun = mRunManager.startNewRun();
                if (mRun == null) {
                    mRun = mRunManager.startNewRun();
                } else {
                    mRunManager.startTrackingRun(mRun);
                }
                updateUI();
            }
        });

        ...
        return view;
    }

    ...

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();
        boolean trackingThisRun = mRunManager.isTrackingRun(mRun);
```

```
        if (mRun != null)
            mStartedTextView.setText(mRun.getStartDate().toString());

        int durationSeconds = 0;
        if (mRun != null && mLastLocation != null) {
            durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
            mLatitudeTextView.setText(Double.toString(mLastLocation.
getLatitude()));
            mLongitudeTextView.setText(Double.toString(mLastLocation.
getLongitude()))
            mAltitudeTextView.setText(Double.toString(mLastLocation.
getAltitude()));
        }
        mDurationTextView.setText(Run.formatDuration(durationSeconds));

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
        mStopButton.setEnabled(started && trackingThisRun);
    }

}
```

Остается оказать еще одну услугу для пользователя приложения: заставить экземпляр `RunFragment` загружать последнее местонахождение текущей серии из базы данных. Реализация будет очень похожа на загрузку `Run`, но с созданием нового объекта `LocationCursor` для работы с объектами `Location`.

Начнем с включения метода для запроса последнего местоположения серии и внутреннего класса `LocationCursor` в `RunDatabaseHelper`.

Листинг 34.24. Запрос последнего местоположения для серии (RunDatabaseHelper.java)

```
public LocationCursor queryLastLocationForRun(long runId) {
    Cursor wrapped = getReadableDatabase().query(TABLE_LOCATION,
            null, // Все столбцы
            COLUMN_LOCATION_RUN_ID + " = ?", // Ограничить заданной серией
            new String[]{ String.valueOf(runId) },
            null, // group by
            null, // having
            COLUMN_LOCATION_TIMESTAMP + " desc", // Сначала самые новые
            "1"); // limit 1
    return new LocationCursor(wrapped);
}

// ... После RunCursor ...

public static class LocationCursor extends CursorWrapper {

    public LocationCursor(Cursor c) {
        super(c);
    }

    public Location getLocation() {
        if (isBeforeFirst() || isAfterLast())
            return null;
        // Сначала получаем поставщика для использования конструктора
```

продолжение ⇨

Листинг 34.24 (*продолжение*)

```
            String provider = getString(getColumnIndex(COLUMN_LOCATION_PROVIDER));
            Location loc = new Location(provider);
            // Заполнение остальных свойств
                loc.setLongitude(getDouble(getColumnIndex(COLUMN_LOCATION_LONGITUDE)));
                loc.setLatitude(getDouble(getColumnIndex(COLUMN_LOCATION_LATITUDE)));
                loc.setAltitude(getDouble(getColumnIndex(COLUMN_LOCATION_ALTITUDE)));
            loc.setTime(getLong(getColumnIndex(COLUMN_LOCATION_TIMESTAMP)));
            return loc;
        }
    }
```

Объект `LocationCursor` выполняет те же функции, что и `RunCursor`, но в нем упаковывается курсор для возвращения строк таблицы `location`, а их различные поля преобразуются в свойства объекта `Location`. В этой реализации есть одна тонкость: конструктору `Location` требуется имя поставщика данных, поэтому мы извлекаем его из текущей строки, прежде чем задавать остальные свойства.

Метод `queryLastLocationForRun(long)` очень похож на `queryRun(long)`, за исключением того, что он ищет последнее местоположение для заданной серии и упаковывает результат в `LocationCursor`.

Как и в случае с `queryRun(long)`, мы должны создать в `RunManager` метод для его вызова и вернуть `Location` из единственной строки курсора.

Листинг 34.25. Получение последнего местоположения для серии (RunManager.java)

```
public Location getLastLocationForRun(long runId) {
    Location location = null;
    LocationCursor cursor = mHelper.queryLastLocationForRun(runId);
    cursor.moveToFirst();
    // Если набор не пуст, получить местоположение
    if (!cursor.isAfterLast())
        location = cursor.getLocation();
    cursor.close();
    return location;
}
```

Теперь новый метод `RunFragment` может использоваться для выборки последнего местоположения текущей серии при создании фрагмента.

Листинг 34.26. Получение последнего местоположения для текущей серии (RunFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    mRunManager = RunManager.get(getActivity());

    // Проверить идентификатор Run и получить объект серии
    Bundle args = getArguments();
    if (args != null) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        if (runId != -1) {
```

```
            mRun = mRunManager.getRun(runId);
            mLastLocation = mRunManager.getLastLocationForRun(runId);
        }
    }
}
```

В результате наш объект RunTracker способен создавать и отслеживать столько серий, сколько выдержит диск (и батарея) вашего устройства. Все эти данные выводятся на экран в логичном, последовательном представлении. Приятного отслеживания!

Упражнение. Выделение текущей серии

В текущей реализации опознать текущую серию можно только одним способом: вручную открыть ее в списке и проверить состояние кнопок запуска и остановки. Было бы удобнее, если бы пользователь мог быстрее и проще обратиться к текущей серии.

Простое упражнение: обеспечьте визуальное выделение строки списка, соответствующей текущей серии (например, выведите рядом с ней значок или измените ее цвет).

Чтобы усложнить задачу, используйте непрерывное оповещение о включенном режиме отслеживания, при нажатии на котором должна запускаться активность RunActivity.

35 Асинхронная загрузка данных

В главе 34 мы реализовали хранение данных с использованием SQLite и использовали объекты Cursor в главном потоке приложения. Однако такое решение нельзя признать полноценным; операции с базой данных следует по возможности вынести из главного потока приложения.

В этой главе мы используем объекты Loader для выборки из базы данных в фоновом потоке. Интерфейс Loader был введен в Android 3.0 (Honeycomb); он также доступен в библиотеке поддержки, поэтому ничто не препятствует его использованию в современных приложениях.

Loader и LoaderManager

Загрузчик (loader) предназначен для загрузки некоторых данных (объекта) из источника. Источником может быть диск, база данных, ContentProvider, сеть или другой процесс. Загрузчик производит выборку данных без блокировки главного потока и доставляет результаты стороне, которая в них заинтересована.

Существуют три встроенных типа загрузчиков: Loader, AsyncTaskLoader и CursorLoader (рис. 35.1). Loader — базовый класс, который сам по себе не очень полезен. Он определяет API, используемый LoaderManager для взаимодействия со всеми загрузчиками.

AsyncTaskLoader — абстрактный класс Loader, использующий AsyncTask для передачи работы в другой поток. Почти все полезные классы загрузчиков, которые вы будете создавать, будут представлять собой субклассы AsyncTaskLoader.

Наконец, CursorLoader расширяет AsyncTaskLoader для загрузки Cursor из ContentProvider через ContentResolver. К сожалению, в приложении RunTracker не

существует возможности использования CursorLoader с курсорами, полученными от SQLiteDatabase.

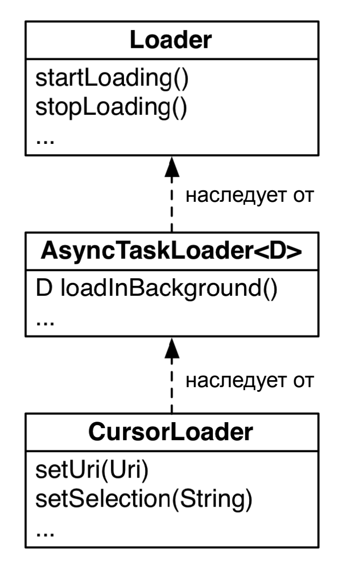


Рис. 35.1. Иерархия классов Loader

Все взаимодействие с загрузчиками осуществляется классом LoaderManager. Этот класс отвечает за запуск, остановку и сопровождение жизненного цикла всех загрузчиков, связанных с вашим компонентом. В пределах Fragment или Activity для получения экземпляра LoaderManager используется метод getLoaderManager().

Метод initLoader(int, Bundle, LoaderCallbacks<D>) запускает инициализацию Loader. В первом аргументе передается целочисленный идентификатор загрузчика, во втором — объект Bundle с аргументами (или null), а последний аргумент содержит реализацию интерфейса LoaderCallbacks<D>.

Как вы увидите в следующих разделах, существует много способов реализации LoaderCallbacks, но чаще всего они реализуются непосредственно в Fragment.

Метод restartLoader(int, Bundle, LoaderCallbacks<D>) выполняет принудительный перезапуск существующего загрузчика. Обычно перезапуск используется для перезагрузки заведомо (или потенциально) устаревших данных.

Интерфейс LoaderCallbacks<D> состоит из трех методов: onCreateLoader(…), onLoadFinished(…) и onLoaderReset(…). Все три метода будут подробно рассмотрены в ходе их реализации в RunTracker.

Почему стоит использовать загрузчик вместо, допустим, прямого использования AsyncTask? Самая убедительная причина заключается в том, что LoaderManager обеспечивает жизнеспособность загрузчиков компонента вместе с их данными при изменениях конфигурации (например, при поворотах).

Если вы будете использовать AsyncTask для загрузки данных, вам придется управлять жизненным циклом объектов при изменениях конфигурации и сохранять данные в месте, которое эти изменения переживет. Часто задача упрощается использованием вызова setRetainInstance(true) для Fragment, и все же в отдельных ситуациях разработчику приходится вмешиваться в происходящее и писать код, который обеспечивает правильность выполнения всего происходящего.

Загрузчики в основном (хотя и не полностью!) избавляют вас от этих хлопот. Если при изменении конфигурации вы инициализируете загрузчик, который уже завершил загрузку своих данных, он может выдать эти данные немедленно, не пытаясь загрузить их заново. Данный механизм работает независимо от того, был ваш фрагмент сохранен (retained) или нет; это упрощает вашу работу, так как вам не приходится учитывать дополнительные аспекты жизненного цикла, обусловленные сохранением фрагментов.

Использование загрузчиков в RunTracker

В настоящее время RunTracker загружает три блока данных: список серий (RunCursor), данные отдельной серии (Run) и последнее местоположение серии (Location). Все данные читаются из базы данных SQLite, поэтому все операции выборки следует переместить в Loader для повышения эффективности работы пользовательского интерфейса.

В следующих разделах мы создадим два абстрактных субкласса AsyncTaskLoader. Первый, SQLiteCursorLoader, представляет собой упрощенную версию системного класса CursorLoader, который работает с объектами Cursor, поступившими из любого источника. Второй, DataLoader<D>, способен загружать произвольные данные; он упрощает использование AsyncTaskLoader для субклассов.

Загрузка списка серий

Текущая реализация RunListFragment напрямую запрашивает у RunManager объект RunCursor, представляющий список серий, в onCreate(Bundle). В этом разделе мы напишем загрузчик, который будет опосредованно выполнять этот запрос в другом потоке. RunListFragment приказывает LoaderManager запускать (и перезапускать) загрузчик и реализует интерфейс LoaderCallbacks для получения информации о готовности данных.

Чтобы упростить код RunListFragment (и классов, которые будут написаны позднее), начнем с создания абстрактного субкласса AsyncTaskLoader с именем SQLiteCursorLoader (листинг 35.1). Этот класс повторяет большую часть кода CursorLoader, но без обязательного использования ContentProvider.

Листинг 35.1. Загрузчик для курсоров SQLite (SQLiteCursorLoader.java)

```
public abstract class SQLiteCursorLoader extends AsyncTaskLoader<Cursor> {
    private Cursor mCursor;

    public SQLiteCursorLoader(Context context) {
        super(context);
    }

    protected abstract Cursor loadCursor();

    @Override
    public Cursor loadInBackground() {
        Cursor cursor = loadCursor();
        if (cursor != null) {
            // Проверить, что окно контента заполнено
            cursor.getCount();
        }
        return cursor;
    }

    @Override
    public void deliverResult(Cursor data) {
```

```
        Cursor oldCursor = mCursor;
        mCursor = data;

        if (isStarted()) {
            super.deliverResult(data);
        }

        if (oldCursor != null && oldCursor != data && !oldCursor.isClosed()) {
            oldCursor.close();
        }
    }

    @Override
    protected void onStartLoading() {
        if (mCursor != null) {
            deliverResult(mCursor);
        }
        if (takeContentChanged() || mCursor == null) {
            forceLoad();
        }
    }

    @Override
    protected void onStopLoading() {
        // Попытаться отменить текущую задачу загрузки, если возможно.
        cancelLoad();
    }

    @Override
    public void onCanceled(Cursor cursor) {
        if (cursor != null && !cursor.isClosed()) {
            cursor.close();
        }
    }

    @Override
    protected void onReset() {
        super.onReset();

        // Убедиться в том, что загрузчик остановлен
        onStopLoading();

        if (mCursor != null && !mCursor.isClosed()) {
            mCursor.close();
        }
        mCursor = null;
    }
}
```

`SQLiteCursorLoader` реализует `AsyncTaskLoader` API для эффективной загрузки и хранения `Cursor` в поле `mCursor`. Метод `loadInBackground()` вызывает абстрактный метод `loadCursor()` для получения `Cursor`, а затем метод `getCount()` для курсора, чтобы убедиться в доступности данных в памяти после их передачи главному потоку.

Метод `deliverResult(Cursor)` решает две задачи. Если загрузчик запущен (это означает, что данные могут быть поставлены), вызывается реализация `deliverResult(…)`

суперкласса. Если старый курсор больше не нужен, он закрывается для освобождения ресурсов. Поскольку существующий курсор может быть кэширован и использован заново, очень важно перед закрытием старого курсора убедиться в том, что старый и новый курсоры различны.

Остальные реализации методов не критичны для понимания смысла RunTracker, но вы можете найти дополнительную информацию в документации API по `AsyncTaskLoader`. При наличии этого базового класса можно реализовать очень простой субкласс `RunListCursorLoader` в `RunListFragment` в виде внутреннего класса.

Листинг 35.2. Реализация RunListCursorLoader (RunListFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Аргумент id содержит идентификатор серии;
    // CursorAdapter автоматически предоставляет эту информацию.
    Intent i = new Intent(getActivity(), RunActivity.class);
    i.putExtra(RunActivity.EXTRA_RUN_ID, id);
    startActivity(i);
}

private static class RunListCursorLoader extends SQLiteCursorLoader {
    public RunListCursorLoader(Context context) {
        super(context);
    }
    @Override
    protected Cursor loadCursor() {
        // Запрос на получение списка серий
        return RunManager.get(getContext()).queryRuns();
    }

}

private static class RunCursorAdapter extends CursorAdapter {
```

Теперь мы можем обновить `RunListFragment` и реализовать интерфейс `LoaderCallbacks` для `Cursor`. Добавьте приведенные ниже методы и включите обратные вызовы в объявление класса.

Листинг 35.3. Реализация LoaderCallbacks<Cursor> (RunListFragment.java)

```
public class RunListFragment extends ListFragment implements LoaderCallbacks<Cursor>
{

    ...

    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        return new RunListCursorLoader(getActivity());
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        // Создание адаптера, ссылающегося на этот курсор
        RunCursorAdapter adapter =
```

```
            new RunCursorAdapter(getActivity(), (RunCursor)cursor);
        setListAdapter(adapter);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Прекращение использования курсора (через адаптер)
        setListAdapter(null);
    }
```

Метод `onCreateLoader(int, Bundle)` вызывается объектом `LoaderManager`, когда ему потребуется создать загрузчика. Аргумент `id` полезен при наличии нескольких однотипных загрузчиков, которых необходимо различать, а объект `Bundle` содержит все передаваемые аргументы. В этой реализации никакие аргументы не используются; она просто создает новый экземпляр `RunListCursorLoader`, ссылающийся на текущий экземпляр `Activity` для контекста.

Метод `onLoadFinished(Loader<Cursor>, Cursor)` будет вызван для главного потока после того, как данные будут загружены в фоновом режиме. В этой версии мы назначаем адаптером `ListView` объект `RunCursorAdapter`, ссылающийся на новый курсор.

Наконец, метод `onLoaderReset(Loader<Cursor>)` будет вызываться при отсутствии доступных данных. Чтобы исключить возможные проблемы, мы прекращаем использовать курсор, присваивая адаптеру списка `null`.

Наконец, после реализации обратных вызовов мы можем приказать `LoaderManager` выполнить его работу. Также можно удалить поле `mCursor` и метод `onDestroy()`, в котором эта переменная очищалась.

Листинг 35.4. Использование Loader (RunListFragment.java)

```
public class RunListFragment extends ListFragment implements LoaderCallbacks<Cursor>
{
    private static final int REQUEST_NEW_RUN = 0;

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        // Запрос на получение списка серий
        mCursor = RunManager.get(getActivity()).queryRuns();
        // Создание адаптера, ссылающегося на этот курсор
        RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);
        setListAdapter(adapter);
        // Инициализация загрузчика для загрузки списка серий
        getLoaderManager().initLoader(0, null, this);
    }

    ...

    @Override
    public void onDestroy() {
        mCursor.close();
```

продолжение ⇗

Листинг 35.4 (*продолжение*)

```
        super.onDestroy();
    }

    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (REQUEST_NEW_RUN == requestCode) {
            mCursor.requery();
            ((RunCursorAdapter)getListAdapter()).notifyDataSetChanged();
            // Перезапуск загрузчика для получения новых серий
            getLoaderManager().restartLoader(0, null, this);
        }
    }
```

После внесения всех изменений запустите приложение и убедитесь в том, что данные в списке заполняются так же, как прежде. При некоторой наблюдательности вы заметите, что во время загрузки данных в фоновом режиме в списке ненадолго появляется вращающийся индикатор прогресса. ListFragment автоматически предоставляет эту функциональность, если адаптеру задается null.

Загрузка одной серии

Класс SQLiteCursorLoader хорошо подходит для загрузки данных, которые должны оставаться в курсоре (например, списков), но в RunFragment мы загружаем два отдельных объекта, а курсор скрывается RunManager. Для работы с произвольными данными нужен более общий загрузчик.

В этом разделе мы создадим новый класс DataLoader, являющийся субклассом AsyncTaskLoader. DataLoader выполняет некоторые простые операции, которые должны выполняться всеми субклассами AsyncTaskLoader, оставляя своим субклассам только реализацию loadInBackground().

Код класса DataLoader приведен в листинге 35.5.

Листинг 35.5. Простой загрузчик данных (DataLoader.java)

```
public abstract class DataLoader<D> extends AsyncTaskLoader<D> {
    private D mData;

    public DataLoader(Context context) {
        super(context);
    }

    @Override
    protected void onStartLoading() {
        if (mData != null) {
            deliverResult(mData);
        } else {
            forceLoad();
        }
    }
```

```
    @Override
    public void deliverResult(D data) {
        mData = data;
        if (isStarted())
            super.deliverResult(data);
    }
}
```

Класс DataLoader использует обобщенный тип D для хранения экземпляра загружаемых данных. В методе onStartLoading() он проверяет доступность этих данных, и если они доступны — немедленно поставляет их. В противном случае он вызывает метод forceLoad() суперкласса для загрузки данных.

Реализация deliverResult(D) сохраняет новый объект данных, и если загрузчик запущен — вызывает реализацию суперкласса для выполнения поставки.

Чтобы использовать новый класс, субклассируйте DataLoader и используйте субкласс RunLoader в RunFragment.

Листинг 35.6. Загрузчик серии (RunLoader.java)

```
public class RunLoader extends DataLoader<Run> {
    private long mRunId;

    public RunLoader(Context context, long runId) {
        super(context);
        mRunId = runId;
    }

    @Override
    public Run loadInBackground() {
        return RunManager.get(getContext()).getRun(mRunId);
    }
}
```

Конструктор RunLoader ожидает получить Context (Activity) и значение long, представляющее идентификатор загружаемой серии. Метод loadInBackground() запрашивает у RunManager серию с заданным идентификатором и возвращает ее.

Когда все будет сделано, вы сможете использовать RunLoader в RunFragment вместо прямого взаимодействия с RunManager в главном потоке. Однако в реализации существует одно отличие: так как RunFragment загружает два разных типа данных, серию и позицию, механизм LoaderCallbacks<D> в сочетании с ограничениями обобщенных типов Java не позволяет напрямую реализовать интерфейс в методах RunFragment. Чтобы обойти ограничения, можно создать внутренние классы, реализующие LoaderCallbacks<D> для Run и Location по отдельности, и передать экземпляр каждого из них при вызове метода initLoader() класса LoaderManager.

Интеграция начинается с добавления внутреннего класса RunLoaderCallbacks в RunFragment.

Листинг 35.7. Класс RunLoaderCallbacks (RunFragment.java)

```
private class RunLoaderCallbacks implements LoaderCallbacks<Run> {

    @Override
    public Loader<Run> onCreateLoader(int id, Bundle args) {
        return new RunLoader(getActivity(), args.getLong(ARG_RUN_ID));
    }

    @Override
    public void onLoadFinished(Loader<Run> loader, Run run) {
        mRun = run;
        updateUI();
    }

    @Override
    public void onLoaderReset(Loader<Run> loader) {
        // Ничего не делать
    }
}
```

В `onCreateLoader(int, Bundle)` возвращается новый экземпляр `RunLoader`, ссылающийся на текущую активность фрагмента, а из пакета аргументов извлекается идентификатор серии. Пакет аргументов передается при вызове `onCreate(Bundle)`.

Реализация `onLoadFinished(…)` сохраняет загруженную серию в поле `mRun` фрагмента и вызывает метод `updateUI()`, чтобы обновленные данные были отражены в пользовательском интерфейсе.

Метод `onLoaderReset(…)` в этом случае не нужен, так как экземпляр `Run` полностью находится в памяти.

Далее используйте реализацию обратных вызовов с `LoaderManager` в методе `onCreate(Bundle)` класса `RunFragment`. Мы также добавим константу `LOAD_RUN`, которая будет использоваться для идентификации загрузчика в наборе `LoaderManager` для `RunFragment`.

Листинг 35.8. Загрузка серии (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_RUN = 0;

    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());

        // Проверка идентификатора серии в аргументе и поиск серии
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
```

```
            if (runId != -1) {
                mRun = mRunManager.getRun(runId);
                LoaderManager lm = getLoaderManager();
                lm.initLoader(LOAD_RUN, args, new RunLoaderCallbacks());
            }
        }
    }
```

Снова запустите RunTracker и убедитесь в том, что приложение работает, как прежде, но на этот раз данные серии загружаются в другом потоке. Если вы очень наблюдательны (или ваш эмулятор работает очень медленно), возможно, вы увидите, как пользовательский интерфейс сначала заполняется без даты серии, а затем обновляется.

Загрузка последней позиции в серии

На последнем шаге мы выведем загрузку последней позиции из главного потока. Эта работа почти идентична той, которую мы только что выполнили для загрузки серии, разве что данными на этот раз будет последняя позиция серии.

Начнем с создания класса `LastLocationLoader`, который будет выполнять непосредственную работу.

Листинг 35.9. Класс LastLocationLoader (LastLocationLoader.java)

```
public class LastLocationLoader extends DataLoader<Location> {
    private long mRunId;

    public LastLocationLoader(Context context, long runId) {
        super(context);
        mRunId = runId;
    }
    @Override
    public Location loadInBackground() {
        return RunManager.get(getContext()).getLastLocationForRun(mRunId);
    }
}
```

Этот класс почти идентичен `RunLoader`, не считая того, что он вызывает метод `getLastLocationForRun(long)` класса `RunManager` с идентификатором серии.

Затем реализуйте внутренний класс `LocationLoaderCallbacks` в `RunFragment`.

Листинг 35.10. Класс LocationLoaderCallbacks (RunFragment.java)

```
private class LocationLoaderCallbacks implements LoaderCallbacks<Location> {

    @Override
    public Loader<Location> onCreateLoader(int id, Bundle args) {
        return new LastLocationLoader(getActivity(), args.getLong(ARG_RUN_ID));
    }

    @Override
    public void onLoadFinished(Loader<Location> loader, Location location) {
```

продолжение ⇗

Листинг 35.10 (*продолжение*)

```
            mLastLocation = location;
            updateUI();
        }

        @Override
        public void onLoaderReset(Loader<Location> loader) {
            // Ничего не делать
        }
    }
```

И снова класс почти идентичен `RunLoaderCallbacks`, не считая того, что он обновляет поле `mLastLocation` перед обновлением пользовательского интерфейса. Далее остается лишь использовать новый загрузчик вместо прямого вызова `RunManager` в `onCreate(Bundle)`, указывая новый идентификатор загрузчика `ID LOAD_LOCATION`.

Листинг 35.11. Загрузка данных последнего местоположения (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_RUN = 0;
    private static final int LOAD_LOCATION = 1;
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());

        // Проверка идентификатора серии в аргументе и поиск серии
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            if (runId != -1) {
                LoaderManager lm = getLoaderManager();
                lm.initLoader(LOAD_RUN, args, new RunLoaderCallbacks());
                mLastLocation = mRunManager.getLastLocationForRun(runId);
                lm.initLoader(LOAD_LOCATION, args, new LocationLoaderCallbacks());
            }
        }
    }
```

Теперь приложение RunTracker может загружать все свои основные данные в фоновом потоке благодаря применению загрузчиков. Запустите его и убедитесь в том, что все работает так же, как прежде.

36 Карты

Следующим логичным шагом RunTracker станет отображение карты перемещений пользователя. При использовании Google Maps API (версии 2) это совсем несложно. В этой главе мы создадим новый класс `RunMapFragment`, который будет выводить карту перемещений пользователя и интерактивные маркеры, обозначающие начало и конец перемещения.

Впрочем, прежде чем браться за самое интересное, необходимо настроить проект для использования Maps API.

Добавление Maps API в приложение RunTracker

Maps API (версия 2) предоставляется пакетом Google Play services SDK и предъявляет ряд требований как к среде разработки, так и к приложению.

Тестирование на реальном устройстве

Пакет Google Play services SDK (а следовательно, и Maps API) требует реального устройства с Android версии не менее 2.2 с установленной поддержкой магазина Google Play. Запуск на эмуляторе не поддерживается.

Установка и использование Google Play services SDK

Чтобы поддержка Maps API стала доступной в вашем проекте, сначала необходимо установить дополнение Google Play services SDK и настроить проект библиотеки для работы с приложением. Следующее описание поможет вам в этом, но самая свежая информация всегда доступна по адресу *http://developer.android.com/google/play-services/*.

1. В Android SDK Manager установите дополнение Google Play services из раздела Extras. Дополнение будет установлено в подкаталог extras/google/google_play_services каталога Android SDK.
2. В Eclipse импортируйте *копию* проекта библиотеки в рабочее пространство командой File ▸ Import... ▸ Existing Android Code Into Workspace. Проект библиотеки находится в каталоге дополнений Google Play services libproject/google-play-services_lib. Обязательно выберите режим копирования Copy projects into workspace в мастере импортирования, чтобы работать с самостоятельной копией проекта.
3. Откройте окно свойств проекта RunTracker и добавьте ссылку на проект библиотеки в категории Android, Library. Щелкните на кнопке Add... и выберите проект google-play-services_lib.

Получение ключа Google Maps API

Чтобы использовать Maps API, необходимо создать ключ API для вашего приложения. Этот процесс состоит из нескольких шагов, и документация Google лучше любого описания, которое мы можем предоставить здесь. Обратитесь по адресу *https://developers.google.com/maps/documentation/android/start* и выполните инструкции для получения собственного ключа.

Обновление манифеста RunTracker

Для работы Google Play services и Maps API необходимо включить в манифест приложения несколько разрешений и требований (в дополнение к только что полученному вами ключу API). Добавьте выделенную разметку XML в манифест RunTracker. Будьте внимательны и проследите за тем, чтобы разрешение с окончанием MAPS_RECEIVE начиналось с имени пакета RunTracker.

Заодно добавьте запись для будущей активности RunMapActivity.

Листинг 36.1. Требования Maps API (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.runtracker"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />

  <permission
    android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>
  <uses-permission
    android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"/>

  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

```
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

  <uses-feature android:required="true"
    android:name="android.hardware.location.gps" />
  <uses-feature
    android:required="true"
    android:glEsVersion="0x00020000"/>
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".RunListActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <activity android:name=".RunActivity"
      android:label="@string/app_name" />
    <activity android:name=".RunMapActivity"
      android:label="@string/app_name" />
    <receiver android:name=".TrackingLocationReceiver"
      android:exported="false">
      <intent-filter>
        <action
          android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
       </intent-filter>
    </receiver>
    <meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value="your-maps-API-key-here"/>
  </application>

</manifest>
```

Вывод местоположения пользователя на карте

После выполнения всех требований пришло время воспользоваться плодами работы и вывести карту. Maps API включает классы `MapFragment` и `SupportMapFragment`, которые субклассируются для настройки представления `MapView` и связанного с ним объекта модели `GoogleMap`.

Создайте субкласс `SupportMapFragment` с именем `RunMapFragment` в новом файле, код которого приведен в листинге 36.2.

Листинг 36.2. Базовая реализация RunMapFragment (RunMapFragment.java)

```
public class RunMapFragment extends SupportMapFragment {
    private static final String ARG_RUN_ID = "RUN_ID";

    private GoogleMap mGoogleMap;
```

продолжение ⇨

Листинг 36.2 (*продолжение*)

```
    public static RunMapFragment newInstance(long runId) {
        Bundle args = new Bundle();
        args.putLong(ARG_RUN_ID, runId);
        RunMapFragment rf = new RunMapFragment();
        rf.setArguments(args);
        return rf;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = super.onCreateView(inflater, parent, savedInstanceState);

        // Сохранение ссылки на GoogleMap
        mGoogleMap = getMap();
        // Вывод местоположения пользователя
        mGoogleMap.setMyLocationEnabled(true);

        return v;
    }

}
```

Метод `newInstance(long)` класса `RunMapFragment` получает идентификатор серии и сохраняет в аргументах новый экземпляр фрагмента, как это делалось в коде `RunFragment`. Вскоре мы используем этот аргумент при выборке списка позиций.

Реализация `onCreateView(…)` вызывает реализацию суперкласса для получения и возвращения представления, но ее работа основана на том факте, что этот вызов инициализирует экземпляр `GoogleMap` фрагмента. `GoogleMap` — объект модели, привязанный к `MapView`, который будет использоваться для настройки различных дополнений к карте. В исходной версии фрагмента просто вызовите `setMyLocationEnabled(boolean)`, чтобы пользователь мог видеть свою позицию на карте и перемещаться к ней.

Для размещения фрагмента `RunMapFragment` создайте простой класс `RunMapActivity` с использованием приведенного ниже кода. Не забудьте, что ссылка на этот класс уже была включена в манифест.

Листинг 36.3. Активность-хост для фрагмента (RunMapActivity.java)

```
public class RunMapActivity extends SingleFragmentActivity {
    /** Ключ для передачи идентификатора серии в формате long */
    public static final String EXTRA_RUN_ID =
        "com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment() {
        long runId = getIntent().getLongExtra(EXTRA_RUN_ID, -1);
        if (runId != -1) {
            return RunMapFragment.newInstance(runId);
        } else {
            return new RunMapFragment();
        }
    }
}
```

Теперь нам понадобится код запуска RunMapActivity из RunFragment при доступности серии. Для этого в макет будет добавлена кнопка, которая будет вызывать карту. Но сначала, как обычно, включим несколько строк, которые будут использоваться кнопкой, а также другими компонентами пользовательского интерфейса в оставшейся части этой главы.

Листинг 36.4. Строки для пользовательского интерфейса (res/values/strings.xml)

```
  <string name="new_run">New Run</string>
  <string name="map">Map</string>
  <string name="run_start">Run Start</string>
  <string name="run_started_at_format">Run started at %s</string>
  <string name="run_finish">Run Finish</string>
  <string name="run_finished_at_format">Run finished at %s</string>
</resources>
```

Внесите изменения в макет RunFragment и включите в него новую кнопку Map.

Листинг 36.5. Добавление кнопки Map (fragment_run.xml)

```
    <Button android:id="@+id/run_stopButton"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/stop"
      />
    <Button android:id="@+id/run_mapButton"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/map"
      />
  </LinearLayout>
</TableLayout>
```

Теперь включите в RunFragment код поддержки новой кнопки и управления ее доступностью.

Листинг 36.6. Подключение кнопки Map (RunFragment.java)

```
public class RunFragment extends Fragment {
    ...

    private RunManager mRunManager;

    private Run mRun;
    private Location mLastLocation;

    private Button mStartButton, mStopButton, mMapButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
```

продолжение ⇨

Листинг 36.6 (*продолжение*)

```
        ...

        mMapButton = (Button)view.findViewById(R.id.run_mapButton);
        mMapButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), RunMapActivity.class);
                i.putExtra(RunMapActivity.EXTRA_RUN_ID, mRun.getId());
                startActivity(i);
            }
        });
        updateUI();

        return view;
    }

    ...

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();
        boolean trackingThisRun = mRunManager.isTrackingRun(mRun);

        if (mRun != null)
            mStartedTextView.setText(mRun.getStartDate().toString());

        int durationSeconds = 0;
        if (mRun != null && mLastLocation != null) {
            durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
            mLatitudeTextView.setText(Double.toString
                                     (mLastLocation.getLatitude()));
            mLongitudeTextView.setText(Double.toString
                                      (mLastLocation.getLongitude()));
            mAltitudeTextView.setText(Double.toString
                                     (mLastLocation.getAltitude()));
            mMapButton.setEnabled(true);
        } else {
            mMapButton.setEnabled(false);
        }
        mDurationTextView.setText(Run.formatDuration(durationSeconds));

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started && trackingThisRun);
    }
```

В новой версии приложение RunTracker сможет вывести карту и ваше местоположение на ней. Запустите приложение, загрузите серию и нажмите кнопку Map. Примерный вид приложения показан на рис. 36.1, конечно, если у вас имеется доступ к Интернету и вы находитесь где-то неподалеку от офиса Big Nerd Ranch.

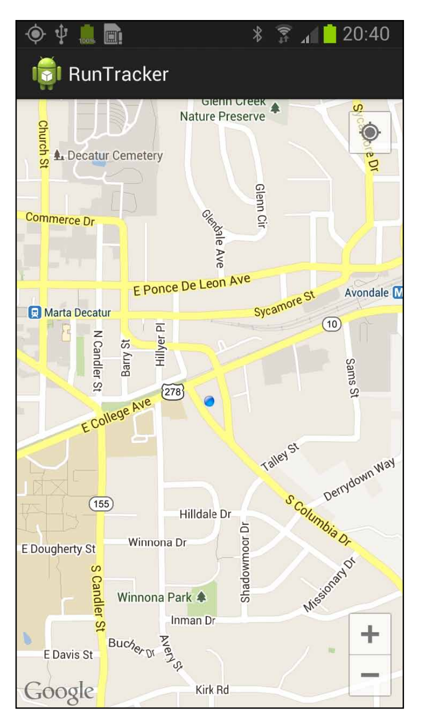


Рис. 36.1. Примерный вид приложения RunTracker

Вывод маршрута

Наша следующая задача — вывод линии, изображающей маршрут. Благодаря Maps API эта задача решается тривиально, но сначала необходимо получить список позиций для построения маршрута. Включите в классы `RunDatabaseHelper` и `RunManager` метод, предоставляющий объект `LocationCursor` с необходимыми данными.

Листинг 36.7. Запрос позиций серии (RunDatabaseHelper.java)

```
public LocationCursor queryLocationsForRun(long runId) {
    Cursor wrapped = getReadableDatabase().query(TABLE_LOCATION,
            null,
            COLUMN_LOCATION_RUN_ID + " = ?", // Отбор по заданной серии
            new String[]{ String.valueOf(runId) },
            null, // group by
            null, // having
            COLUMN_LOCATION_TIMESTAMP + " asc"); // Упорядочить
    return new LocationCursor(wrapped);          // по временной метке
}
```

Метод `queryLocationsForRun(long)` очень похож на метод `queryLastLocationForRun(long)` из главы, посвященной SQLite, но он упорядочивает позиции по возрастанию, и возвращает их все.

Используя этот метод в RunManager, можно создать удобный *фасад* (façade) для RunMapFragment.

Листинг 36.8. Запрос позиций серии, часть II (RunManager.java)

```
public LocationCursor queryLocationsForRun(long runId) {
    return mHelper.queryLocationsForRun(runId);
}
```

RunMapFragment может использовать этот новый метод для загрузки позиций. Естественно, запрос к базе данных следует вынести из главного потока при помощи Loader. Создайте новый класс LocationListCursorLoader для решения этой задачи.

Листинг 36.9. Загрузчик позиций (LocationListCursorLoader.java)

```
public class LocationListCursorLoader extends SQLiteCursorLoader {
    private long mRunId;

    public LocationListCursorLoader(Context c, long runId) {
        super(c);
        mRunId = runId;
    }

    @Override
    protected Cursor loadCursor() {
        return RunManager.get(getContext()).queryLocationsForRun(mRunId);
    }
}
```

Используйте новый загрузчик в RunMapFragment для загрузки позиций.

Листинг 36.10. Загрузка позиций в RunMapFragment (RunMapFragment.java)

```
public class RunMapFragment extends SupportMapFragment
    implements LoaderCallbacks<Cursor> {
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_LOCATIONS = 0;

    private GoogleMap mGoogleMap;
    private LocationCursor mLocationCursor;

    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Проверка идентификатора серии в аргументе и поиск серии
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            if (runId != -1) {
                LoaderManager lm = getLoaderManager();
                lm.initLoader(LOAD_LOCATIONS, args, this);
            }
        }
    }
```

```
    ...
    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        return new LocationListCursorLoader(getActivity(), runId);
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        mLocationCursor = (LocationCursor)cursor;
    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Завершение работы с данными
        mLocationCursor.close();
        mLocationCursor = null;
    }
}
```

Класс RunMapFragment сохраняет объект LocationCursor для списка позиций. Этот список будет использоваться при построении карты с маршрутом.

Реализация onLoaderReset(Loader<Cursor>) закрывает и обнуляет ссылку на курсор, когда тот становится недоступным. В обычных обстоятельствах этот метод будет вызываться при закрытии загрузчика объектом LoaderManager при выходе пользователя с фрагмента, содержащего карту. Вдобавок такое решение не закрывает курсор при выполнении поворота, а это именно то, что нам нужно.

Мы подошли к самой интересной части этой главы: выводу информации на GoogleMap. Добавьте метод updateUI() с заполнением данных отображаемой серии, и вызовите этот метод в onLoadFinished(…).

Листинг 36.11. Нанесение данных серии на карту (RunMapFragment.java)

```
private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Создание накладки с позициями серии.
    // Создаем ломаную со всеми точками.
    PolylineOptions line = new PolylineOptions();
    // Также создается объект LatLngBounds для масштабирования по размерам.
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Перебор позиций
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());
        line.add(latLng);
        latLngBuilder.include(latLng);
        mLocationCursor.moveToNext();
    }
    // Добавление маршрута на карту
    mGoogleMap.addPolyline(line);
    // Масштабирование карты по маршруту с дополнительными отступами
```

продолжение ⇗

Листинг 36.11 (*продолжение*)

```
        // В качестве ограничивающего прямоугольника выбирается
        // размер текущего экрана.
        Display display = getActivity().getWindowManager().getDefaultDisplay();
        // Построение инструкции перемещения для камеры карты.
        LatLngBounds latLngBounds = latLngBuilder.build();
        CameraUpdate movement = CameraUpdateFactory.newLatLngBounds(latLngBounds,
                display.getWidth(), display.getHeight(), 15);
        mGoogleMap.moveCamera(movement);
    }

        @Override
        public Loader<Cursor> onCreateLoader(int id, Bundle args) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            return new LocationListCursorLoader(getActivity(), runId);
        }
        @Override
        public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
            mLocationCursor = (LocationCursor)cursor;
            updateUI();
        }
```

Новый метод использует Maps API в нескольких местах. Сначала он создает экземпляр `PolylineOptions`, который будет использоваться для построения маршрута, выводимого на экран, и экземпляр `LatLngBounds.Builder` для создания ограничивающего прямоугольника для масштабирования карты.

Затем он перебирает данные `LocationCursor` и для каждого объекта `Location` создает объект `LatLng` с его координатами. Объект `LatLng` включается в `PolylineOptions`, а последний включается в `LatLngBounds` перед переходом к следующей строке в курсоре.

После перебора всех позиций метод вызывает `addPolyline(PolylineOptions)` для `GoogleMap`, добавляя маршрут на карту.

На следующем шаге следует масштабировать карту так, чтобы на ней помещалась вся линия. Перемещение по карте относится к ответственности «камеры», а для настройки камеры используются команды, упакованные в экземплярах `CameraUpdate` и передаваемые `moveCamera(CameraUpdate)`. Создание экземпляра для перемещения камеры по границам только что добавленной линии выполняется методом `newLatLngBounds(LatLngBounds, int, int, int)` класса `CameraUpdateFactory`.

Мы передаем размеры текущего экрана как приближенное значение размера карты в пикселах, с добавлением нескольких пикселов на отступы, чтобы маршрут лучше смотрелся. Существует упрощенная версия этого метода `newLatLngBounds(LatLngBounds, int)`, но она выдает исключение `IllegalStateException`, если метод будет вызван до того, как `MapView` завершит определение своих размеров в процессе построения макета. Так как мы не можем гарантировать, что это произойдет к моменту вызова `updateUI()`, приходится использовать приблизительную оценку.

Внесите изменения и снова запустите RunTracker — на карте появляется черная линия, обозначающая маршрут. Пожалуй, сейчас стоит поближе познакомиться с классом `PolylineOptions` и попытаться немного украсить результат.

Добавление маркеров начала и конца маршрута

Теперь, когда мы подготовили все необходимое, добавить маркеры, обозначающие начальную и конечную позиции маршрута, относительно несложно. Мы также добавим текст, который будет выводиться в информационном окне при прикосновении к маркеру.

Добавьте в метод `updateUI()` код, выделенный жирным шрифтом.

Листинг 36.12. Маркеры начала и конца маршрута с информацией (RunMapFragment.java)

```
private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Создание накладки с позициями серии.
    // Создаем ломаную со всеми точками.
    PolylineOptions line = new PolylineOptions();
    // Также создается объект LatLngBounds для масштабирования по размерам.
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Перебор позиций
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());

        Resources r = getResources();

        // Если это первая позиция, добавить маркер
        if (mLocationCursor.isFirst()) {
            String startDate = new Date(loc.getTime()).toString();
            MarkerOptions startMarkerOptions = new MarkerOptions()
                .position(latLng)
                .title(r.getString(R.string.run_start))
                .snippet(r.getString(R.string.run_started_at_format, startDate));
            mGoogleMap.addMarker(startMarkerOptions);
        } else if (mLocationCursor.isLast()) {
            // Если это последняя позиция, которая не является
            // также первой, добавить маркер
            String endDate = new Date(loc.getTime()).toString();
            MarkerOptions finishMarkerOptions = new MarkerOptions()
                .position(latLng)
                .title(r.getString(R.string.run_finish))
                .snippet(r.getString(R.string.run_finished_at_format, endDate));
            mGoogleMap.addMarker(finishMarkerOptions);
        }

        line.add(latLng);
        latLngBuilder.include(latLng);
        mLocationCursor.moveToNext();
    }
    // Добавление маршрута на карту
    mGoogleMap.addPolyline(line);
```

Для первой и последней точки маршрута код создает экземпляр `MarkerOptions` для хранения позиции, заголовка и дополнительного текста. Заголовок

и дополнительный текст выводятся в простом информационном окне, когда пользователь прикасается к маркеру.

В этом коде используется маркер по умолчанию, но при желании можно создать маркер другого цвета (и даже содержащий заданное изображение) при помощи метода `icon(BitmapDescriptor)` и `BitmapDescriptorFactory`. Существует много стандартных цветов, из которых вы можете выбрать нужный.

Запустите приложение RunTracker и протестируйте его во время своего следующего путешествия. Счастливого пути!

Упражнение. Оперативное обновление

В текущей версии `RunMapFragment` может отображать карту с местоположениями серии, «замороженными» по времени на момент прибытия пользователя. Качественное геопозиционное приложение должно выводить обновления в режиме «живого» обновления. Используя субкласс `LocationReceiver` в `RunMapFragment`, организуйте обработку поступления новых позиций с перерисовкой маршрута. Не забудьте удалить предыдущую накладку (overlay) и маркеры, прежде чем добавлять новые.

37 Послесловие

Поздравляем! Вы добрались до последней страницы этого учебника. Не каждому хватило бы терпения сделать то, что вы сделали; узнать то, что вы узнали. Ваша самоотверженная работа не пропала даром; теперь вы стали разработчиком Android.

Последнее упражнение

У нас осталось еще одно, последнее упражнение для вас: станьте *хорошим* разработчиком Android. Каждый хороший разработчик хорош по-своему, поэтому с этого момента вы должны найти собственный путь.

С чего начать, спросите вы? Мы можем дать несколько советов.

- *Пишите код.* Начинайте прямо сейчас. Вы быстро забудете то, что узнали в книге, если не будете применять полученные знания. Примите участие в проекте или напишите собственное простое приложение. Что бы вы ни выбрали, не тратьте времени: пишите код.
- *Учитесь.* В этой книге вы узнали много полезной информации. Что-то из этого пробудило ваше воображение? Поэкспериментируйте со своими любимыми возможностями. Найдите и почитайте дополнительную документацию по ним — или целую книгу, если она есть.
- *Встречайтесь с людьми.* Многие первоклассные разработчики Android постоянно бывают на канале `#android-dev` сервера `irc.freenode.net`. Сайт Android Developer Office Hours (*https://plus.google.com/+AndroidDevelopers/posts*) поможет вам оставаться на связи с группой разработки Android и другими заинтересованными разработчиками. Локальные встречи также помогут вам найти единомышленников.
- *Присоединяйтесь к сообществу разработки с открытым кодом.* Разработка Android процветает на сайте *http://www.github.com.* Обнаружив полезную библиотеку, посмотрите, в каких еще проектах участвуют его авторы. Делитесь своим кодом — никогда не знаешь, кому он пригодится.

Бессовестная самореклама

Если вам понравилась книга, ознакомьтесь с другими учебниками Big Nerd Ranch по адресу *http://www.bignerdranch.com/books*. У нас также имеется широкий выбор недельных курсов для разработчиков, на которых вы всего за неделю сможете получить массу полезной информации. И конечно, если вам понадобятся услуги опытных разработчиков, мы также занимаемся контрактным программированием. За подробностями обращайтесь на наш сайт по адресу *http://www.bignerdranch.com*.

Спасибо

Без таких читателей, как вы, наша работа была бы невозможной. Спасибо вам за то, что купили и прочитали нашу книгу.